Ingeniería Técnica Informática de Gestión

PROYECTO FIN DE CARRERA

ESTUDIO DEL STACK TECNOLÓGICO PARA EL DESARROLLO DE UN FRAMEWORK ESCALABLE ORIENTADO A APIS REST

Autor: Giancarlo Alfredo Muñoz Reinoso

Tutor: Alejandro Baldominos Gómez

Leganés, Septiembre 2015

Índice de contenidos

Índice de figuras

1. Introducción

El Software es una de la ultimas consecuencias de la llamada sociedad de la información (Figura 1 - Sociedad de la información). Este siglo, denominado “siglo de la información” debe su nombre a la cantidad de información que el ser humano posee en la actualidad . El software no es más que una herramienta con la que se intenta canalizar, separar, diferenciar, ordenar y almacenar esta información. Desde mediados del siglo XX en las sociedades modernas se ha producido un movimiento de los medios de creación de la información.

La información se empezó a crear a través de medios digitales, con lo que el problema de procesamiento y almacenamiento de esta información derivó en soluciones también digitales conocidas como TIC (Tecnologías de la Información y las Comunicaciones).

Todo esto al fin y al cabo no es mas que información consumida por el ser humano. Según es sabido el 90 % de la información existente en la actualidad ha sido generada en los últimos 2 años (Daily, 2013), estoy hablando de la información que posee el ser humano, no solo de la información generada desde el inicio de la era de las TIC.

Está muy bien tener acceso a tal cantidad de información, pero también estaría muy bien tener fácil acceso a ella. Para ello es necesario el software, recientemente es sabido que el cerebro de una persona mayor no funciona lento por disminución de su capacidad cognitiva si no por la cantidad de información que este almacena, se podría hacer una similitud al periodo cuando el usuario medio debía formatear su ordenador personal debido a lo lento que este funcionaba.

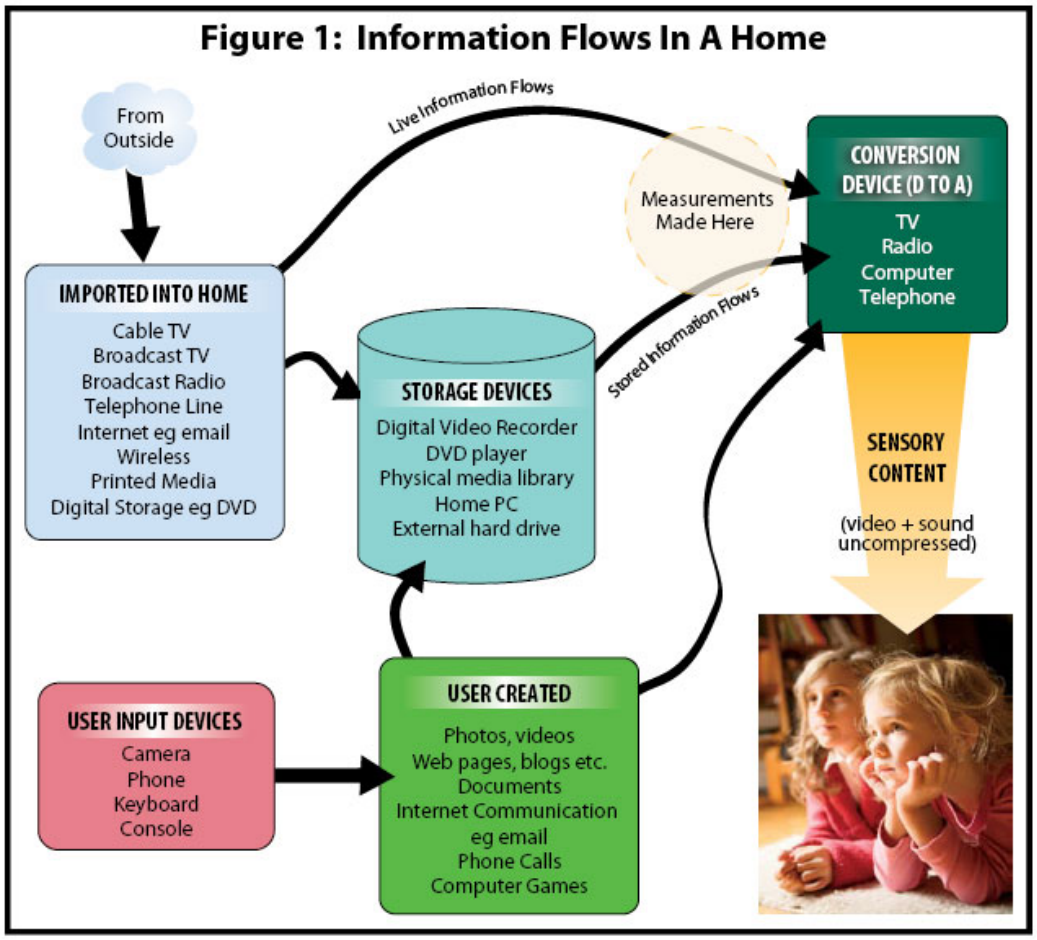


Figura 1 - Sociedad de la información (Beringer)

1.1. Motivación del proyecto

La motivación de tu proyecto es la necesidad de exponer un acceso a la información tanto para consultarla como para modificarla de una forma sencilla, accesible y escalable.

La posibilidad de proveer a otros desarrolladores de un punto de acceso a una plataforma es un problema común en la actualidad con la cantidad de distintos dispositivos que son capaces de generar o consumir información, la necesidad de cierto protocolo común, que sea conciso y ligero es básico.

Es por ello que se creó la necesidad de realizar este concepto de aplicación en el que se aúnan simplicidad, escalabilidad y tolerancia a fallos. La aplicación está totalmente enfocada en ser capaz de responder bajo cualquier concepto (alta latencia, gran cantidad de peticiones, etc.). Para responder a estas demandas incluso la manera de programar debía cambiar, el concepto de escalabilidad requería unos sistemas que pudieran ser tolerantes con un alto nivel de concurrencia y multi-hilo. Por ello es necesario la búsqueda de un entorno de trabajo que sea capaz de tolerar estos requisitos no funcionales de manera sencilla ayudando también al desarrollador a realizar su labor debido a que la programación multi-hilo es altamente compleja de realizar correctamente.

Por ello se han valorado un entorno de trabajo basado en una arquitectura de mensajes asíncronos, mantieniendo ningún hilo en espera, por lo que el sistema es más robusto a fallos y puede ser fácilmente distribuido en varias máquinas.

Debido a la propia experiencia desarrollando aplicaciones contra una API, se dio bastante énfasis a realizar una completa documentación de la funcionalidad disponible para que el programador que vaya a hacer uso de la aplicación sea capaz detener toda la información necesaria.

Una característica importante de este trabajo es que se ha utilizado únicamente software libre, de tal forma que se pueda contribuir con el proyecto al beneficio de la comunidad opensource (/ free

software), realizando así un software que pueda ser útil en el mercado y utilizando tecnologías pioneras dando lugar a un producto de máxima calidad.

1.2. Objetivos

El objetivo de este proyecto es realizar una prueba de concepto con las ultimas tecnologías orientadas a la computación en la nube.

El producto esta enfocado a ofrecer servicios web REST para la gestión de eventos de una manera genérica y proporcionar el código como software libre para la comunidad. De este modo se pretende proveer a los *endpoints* de los servicios web de la información necesaria de las distintas entidades que maneja el modelo de datos así como las acciones que se pueden realizar para modificar estas.

Usando una API, se establece un "contrato" entre los clientes y el servidor de lo que se puede esperar de la aplicación. La aplicación posee una arquitectura sencilla teniendo como principal objetivo mostrar la facilidad con la que se puede llegar a levantar una aplicación moderna (usando únicamente la línea de comandos). Para mostrar la agilidad de la puesta en marcha se usan ciertas tecnologías existentes como Cloud Computing, el framework Spray, etc.

Los principales requisitos del proyecto ha satisfacer han sido:

El servidor correrá sobre la nube

Las ventajas de una infraestructura de hardware remota van desde la nula preocupación por el mantenimiento de esta (costes, actualizaciones, seguridad, etc.) hasta la escalabilidad que

proporciona esta llegado el caso de necesitar una mayor capacidad ya sea de ancho de banda o capacidad de proceso.

Uso y generación de software libre

Por y para el software libre, este es el concepto con el que nace el proyecto. Para ello se creó una primera plantilla del entorno de trabajo donde se usan las principales tecnologías aplicadas, esta plantilla ha sido compartida y presentada dentro de la comunidad de Scala, como un "*seed*" para que otros usuarios puedan reusar y modificar según sus necesidades. (Muñoz Reinoso, SpraySwaggerSlick3Seed, 2015) , (Muñoz Reinoso, Reddit, 2015)

Partiendo de esta plantilla como base se generó el proyecto para la gestión de eventos, además, el código para la gestión de eventos también está disponible públicamente (Muñoz Reinoso, GitHub, 2015)

1.3. Estructura de la memoria

En este capítulo se ha introducido una presentación al proyecto así como los objetivos y motivaciones a realizarlo. En el siguiente capítulo se aborda el estado del arte en las distintas tecnologías, herramientas y metodologías usadas para la realización de este proyecto, tales como: Scala, Spray, Git, Scrum, Intellij y conceptos como la programación funcional.

En el tercer capítulo se explican como se han usado las distintas herramientas en el proyecto y en el cuarto punto se explica el desarrollo del proyecto.

También se incluyen tres anexos, un manual de desarrollador, un manual de usuario y un manual de despliegue.

2. Estado del arte

2.1. Introducción

El *cloud computing* esta en auge, todas las ultimas tendencias en distintos ámbitos de las TI (Tecnologías de la Información) así lo atestiguan, se oye mucho hablar de los distintos servicios y la simplicidad con la que antiguas funciones que se realizaban localmente, ahora pueden ser llevadas a cabo desde un hardware muy básico conectándose a una infraestructura que puede estar a miles de kilómetros de distancia.

La migración a la nube es algo que muchas empresas están llevando a cabo para intentar reducir costes del departamento de sistemas.la computación en la nube es el termino general que envuelve todo lo relacionado con proveer servicios a través de internet.

La computación en la nube tiene tres conceptos fundamentales:

- Tarificado según demanda, es decir solo pagas lo que usas, se acabó el comprar un servidor sobredimensionado a tus necesidades (Figura 2 - Ejemplo de servicio bajo demanda en la nube).

- Facilidad para adaptar la capacidad del servicio según la demanda, por ejemplo, una aplicación web de comercio electrónico que lanza una promoción consiguiendo multiplicar por cien el volumen de usuarios, la solución: el proveedor del servidor en la nube amplia el ancho de banda así como la capacidad de proceso del servidor de manera sencilla.

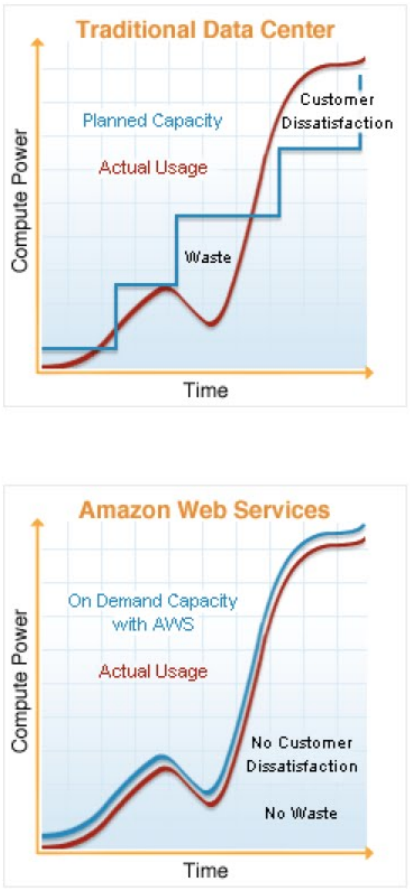


Figura 2 - Ejemplo de servicio bajo demanda en la nube (Ohara, 2009)

- Total abstracción en la administración/instalación/configuración/actualización de los servicios que estas consumiendo, simplemente te preocupas de usarlos, un ejemplo sencillo podría ser *Gmail*, en comparación con Microsoft Outlook, este ultimo tienes que instalarlo (gastando tiempo además de espacio en tu disco duro), actualizarlo y puede que hasta reinstalarlo, mientras que en *Gmail* tu máxima preocupación es que no se te haya olvidado la contraseña.

Lo que se pretende es ofrecer a través de internet una serie de servicios que abarcan desde la capa de hardware o infraestructura (*Infrastucture-as-a-Service*), la capa de plataforma (*Platform-as-a Service*) o la capa de software(*Software-as-a-Service*). Esta separación se basa en el distinto grado de responsabilidad en la administración.

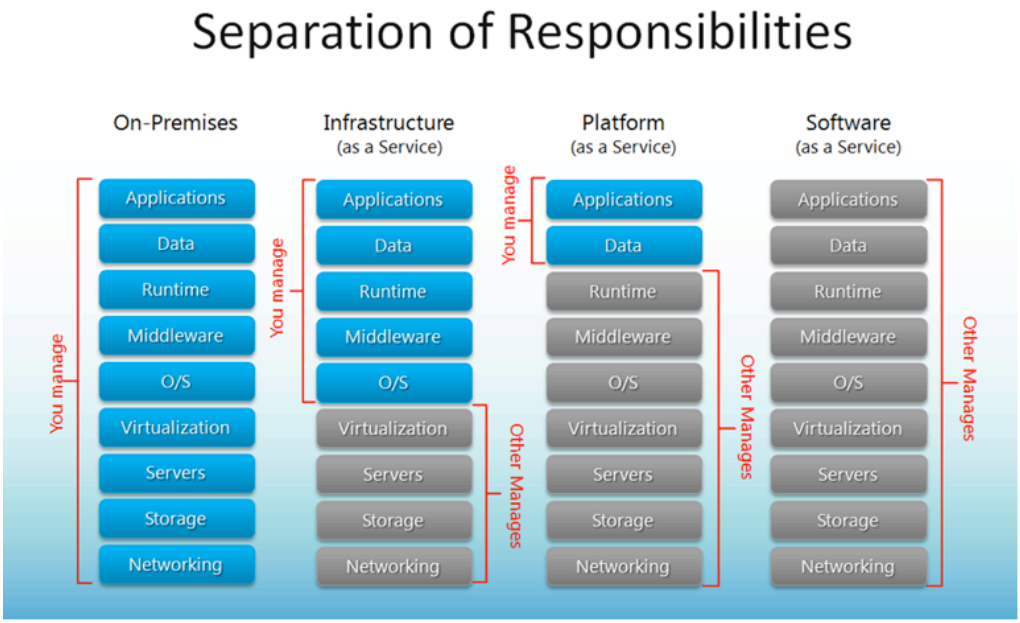


Figura 3 - Separación de responsabilidades en la nube (Remde, 2011)

La responsabilidad en la administración disminuye a medida que subimos hacia capas superiores (Figura 3 - Separación de responsabilidades en la nube). En la Figura 4 - Ejemplos de soluciones en la nube, podemos ver otro ejemplo de la estratificación de las capas así como algunas implementaciones existentes en la actualidad de cada una de ellas.

Figura 4 - Ejemplos de soluciones en la nube (Solutions, 2013)

Podemos comparar la computación en la nube el "*renting*" para los servicios de computación. Por ejemplo para una aplicación como puede ser una tienda de ebooks no necesitas ni tener hardware ni conexión a internet de alta velocidad, simplemente una ventana del navegador.

2.2. Definición del concepto y estructura básica de la PaaS

2.2.1. Introducción

Platform-as-a-Service es la denominación del concepto ofrecido en este caso por Heroku y sobre el que está sustentado la aplicación. En el PaaS el proveedor proporciona el hardware y el software necesario para el desarrollo de la aplicación, por lo que los usuarios de este tipo de servicio se desentienden de tener servidor, instalar software, configurarlo o lanzar la aplicación (Rouse, 2015).

Vendría a ser un paso mas allá del IaaS, solo que en el Paas no habría que preocuparse por la instalación del software mínimo para el despliegue del producto (Sistema operativo, Servidor, etc.)

PaaS permite a pequeñas startups o equipos ser bastante independientes y ofrecer un servicio de calidad y escalable que solo podría ser ofrecido por un equipo especializado consumiendo significantes recursos (tiempo-dinero). Las principales desventajas podrían ser no tener control absoluto sobre datos sensibles, otra desventaja podría ser la discontinuidad del servicio por parte de la plataforma.

El concepto de PaaS permite la realización de prototipos de una manera muy rápida y eficiente lo que permite acelerar la creación de un negocio o hacer un prueba de viabilidad permitiendo que los costes de esta se mantengan muy reducidos (SalesForce, 2014). Por primera vez los desarrolladores son capaces de concentrarse únicamente de construir la aplicación y no administrar complejas infraestructuras software y hardware.

2.2.2. Heroku

Heroku es la plataforma PaaS usada en el proyecto, mediante un simple comando permite desplegar la aplicación en producción. Fue uno de los primeros servicios de este tipo disponibles allá por el año 2007, Esta plataforma además de alojar la aplicación proporciona también el servicio de la base de datos. De manera similar a otros PaaS, Heroku permite que el desarrollador emplee todo su tiempo en el desarrollo del código y no en tareas propias de sistemas.

En la elección de Heroku como soporte para nuestro proyecto se barajo la posibilidad también de otras opciones de PaaS, en este caso Amazon Elastic Beanstalk.

Amazon Elastic Beanstalk ofrece prácticamente la misma funcionalidad que Heroku, es decir con un simple comando seriamos capaces de desplegar nuestra aplicación, el problema es que por el momento no ofrecen soporte para Scala por lo que se decidió a usar Heroku.

2.2.3. Arquitectura de Heroku

Heroku provee el hosting y los servidores además de encargarse del mantenimiento del software, además de una continua mejora de la plataforma. (COLOANDCLOUD, 2015)

La arquitectura esta compuesta por seis capas (Figura 5 - Arquitectura de Heroku):

1. HTTP Reverse Proxy
2. HTTP Cache
3. Routing Mesh
4. Dynos
5. Database
6. Cache

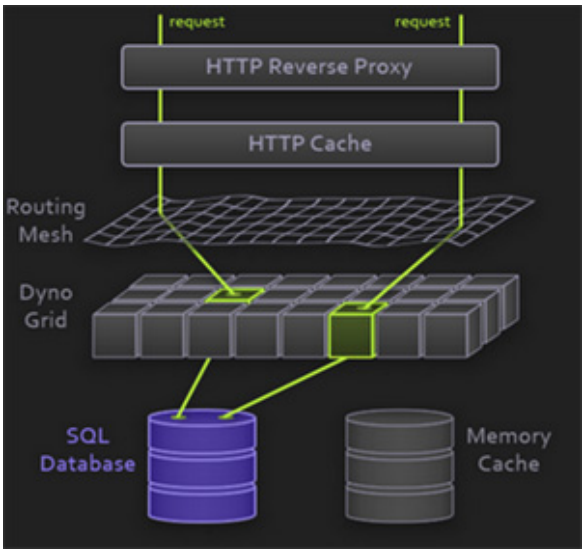


Figura 5 - Arquitectura de Heroku (COLOANDCLOUD, 2015)

"Dyno" (Figura 6 - Heroku Dyno) seria la unidad donde estaría alojada la base de datos. Según la web de heroku, los Dynos son "unidades aisladas y tolerantes a fallos, es un proceso mantenido por la estructura Dyno Manifold, un Dyno recibe una petición desde el la capa de routing, accede a la aplicación que aloja y escribe la respuesta en el Logplex " (Heroku)

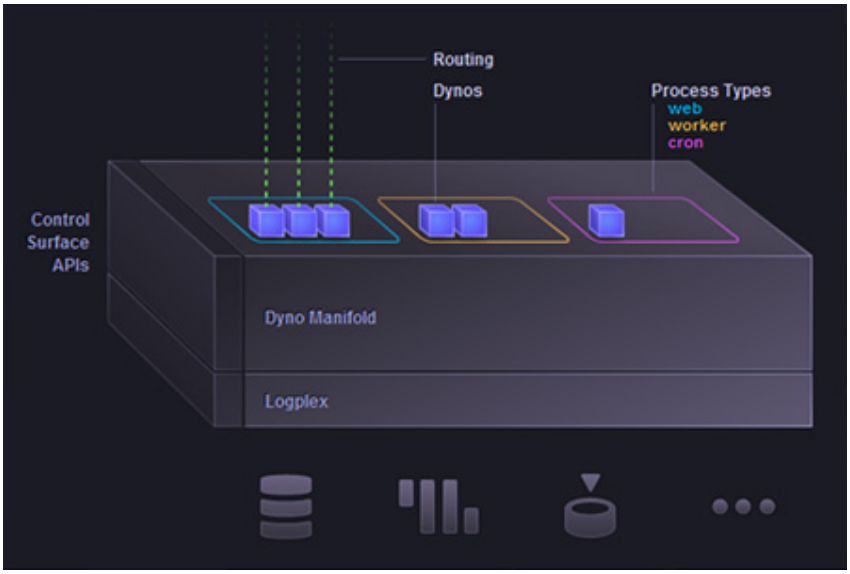


Figura 6 - Heroku Dyno (COLOANDCLOUD, 2015)

2.3. Control de versiones

El control de versiones es un sistema que registra los cambios realizado a un archivo o conjunto de archivos a través del tiempo, de manera que puedes acceder a un determinado momento en el que el fichero fue modificado.

Si eres diseñador gráfico o diseñador web y quieres mantener cada versión de una imagen o de un diseño (que seguramente lo querrás hacer), usar un Sistema de Control de Versiones (VCS por su siglas en inglés) es una sabia decisión. Te permite revertir archivos a un estado previo, revertir el proyecto completo a un estado anterior, comparar cambios en el tiempo, ver quien modificó algo por última vez que pudo estar causando un problema, quien introdujo un problema y cuando, y más. Usar un VCS también significa que si dañas algo o pierdes archivos, generalmente podrás recuperarlos con facilidad. Adicionalmente, obtienes todo esto por muy poco trabajo extra. (Git-SCM)

2.3.1. Git

Es el sistema de control de versiones usado para el desarrollo del proyecto, debido a la naturaleza opensource del proyecto se decidió por Git (Figura 7 - Código del proyecto en), usando como repositorio GitHub (www.github.com).

GitHub es usado por la comunidad opensource como el principal alojamiento para sus proyectos, por ejemplo el lenguaje Scala (https://github.com/scala).

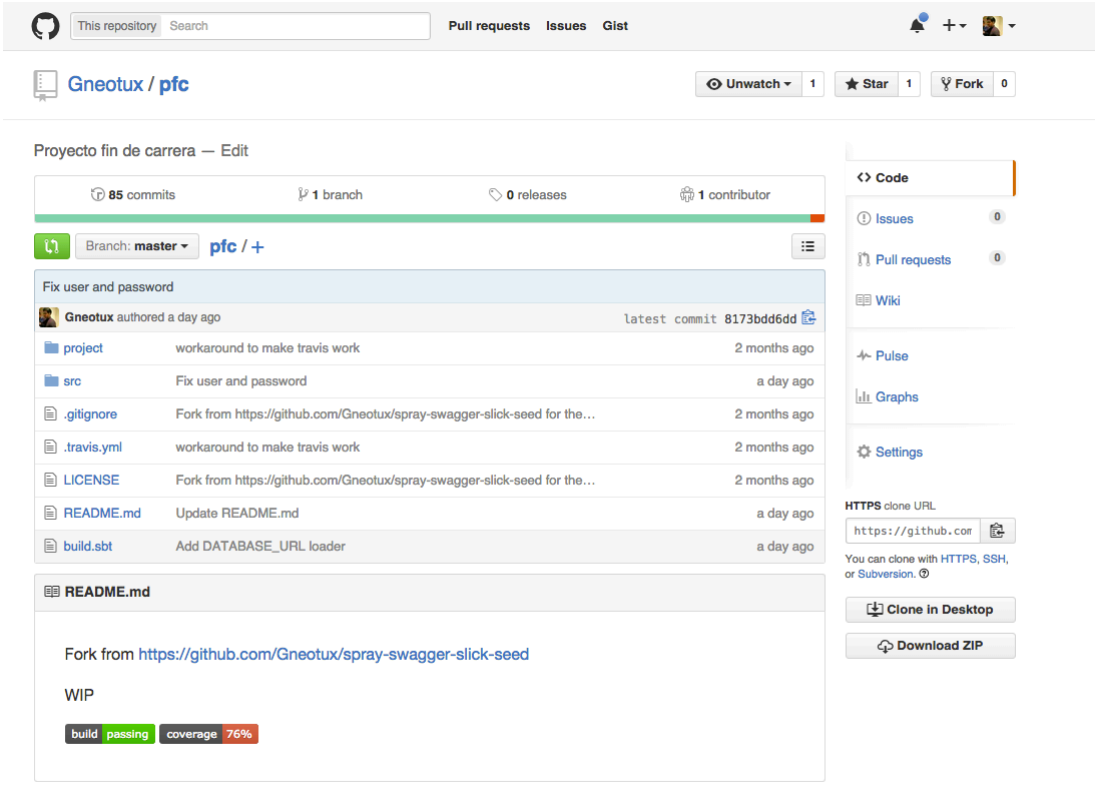


Figura 7 - Código del proyecto en GitHub

Existen muchas alternativas a Git, la mas común y usada podría ser SVN, no se barajo usar SVN debido a la pobre funcionalidad que este CVS ofrece en comparación con Git, además al presentar el proyecto como *OpenSource* y debido a que la casi totalidad de estos están alojados en GitHub usando Git, se decidió por este último.

2.4. Ingeniería de servicios web

2.4.1. Proceso de desarrollo web

En la actualidad, el uso de la red se ha convertido en un elemento indispensable, transformando las necesidades de información de las empresas y organizaciones. No existe prácticamente ninguna empresa u organismo que no necesite que la información esté accesible tanto dentro como fuera de su edificio o que esta sea compartida entre todas las partes interesadas en ella, ya sea en su totalidad o en aquella parte que corresponda según su función.

Estas necesidades han sido las causantes de un movimiento creciente de cambio desde las clásicas aplicaciones de escritorio a aplicaciones Web, que las satisfacen ampliamente. Así, las páginas Web, que antes tan solo se dedicaban a mostrar información, se han convertido en aplicaciones con las cuales el usuario interactúa. No solo esto, además de la clásica comunicación usuario-aplicación ahora existen diferentes dispositivos comunicándose entre si a través de la Web (Smartphones, Tablets, Wearables, etc.).

Desde el punto de vista del desarrollador esto plantea diferentes retos, uno de ellos es el de encontrar la mejor solución para que dos sistemas se comuniquen entre si. Esto llevó a desarrollar distintos protocolos y servicios web, es decir maneras de comunicarse, algo así como la “lengua” en la que van a hablar las máquinas.

Existe además la consideración de transmitir de manera eficaz las posibilidades que tu sistema ofrece para con otros. Esta información que históricamente ha ido en documentos (escritos), ahora existe la posibilidad de ofrecerlos a través de la propia aplicación, con lo que se provee de información actualizada y totalmente fiable acerca de las capacidades del sistema.

Con anterioridad el uso de servicios SOAP estaba muy extendido en el mundo de las aplicaciones *Enterprise,* este protocolo requería complicados sistema de interpretación por parte de los clientes además de que pequeños cambios en el servidor impactaban de forma negativa en los usuarios.

2.4.2. Lenguaje de programación general

El lenguaje elegido para el desarrollo es Scala, es un lenguaje de programación general, tiene soporte de programación funcional con un sistema de tipado muy estricto. El lenguaje es compatible con el Java bytecode por lo que puede correr sobre cualquier plataforma JVM.

La elección del lenguaje se debió a la experiencia que se tenía de él en el ámbito laboral, después de casi 2 años trabajando con el lenguaje se aprecian sus fortalezas como lenguaje de ámbito general, la posibilidad de disponer de todas las librerías Java existentes, o la facilidad con la que con una simples líneas de código se permite hacer algo que con Java costaría el doble.

Por ejemplo en la Figura 8 - Código Java, se observa una función que a partir de una lista de "orders" devuelve los los productos que esta lista contiene, como se puede observar la misma función se realiza en una línea usando un código totalmente inmutable lo que lo transforma en thread-safe.

```
public List<Product> getProducts() {
    List<Product> products = new ArrayList<Product>();
    for (Order order : orders) {
        products.addAll(order.getProducts());
    }
    return products;
}
```

Figura 8 - Código Java

```
def products = orders.flatMap(o => o.products)
```

Figura 9 - Código Scala

2.4.3. Framework web

Spray framework es entorno de trabajo, que usamos como "esqueleto" de la aplicación.

Spray esta compuesto por usa serie de librerías que proporcionan soporte REST/HTTP sobre el conocido framework *Akka,* librería que proporciona unas herramientas para producir un código altamente concurrente y distribuido en la JVM. El framework está construido completamente usando un lenguaje asíncrono, esto se consigue basando la plataforma en un sistema de mensajes y el uso de *Futures,* objeto dentro de Scala que permite desarrollar un código paralelizable y concurrente.

La arquitectura a grandes rasgos del framework se muestra en la Figura 10 - Arquitectura REST.

Se eligió Spray dentro de las posibles opciones a la hora de crear una api REST estas opciones eran: Play Framework o Scalatra. Esta decisión fue en gran parte por la necesidad de explorar un framework totalmente asíncrono y basado en Akka, desde el punto de vista de los objetivos del proyecto, Spray nos proporciona esa escalabilidad y tolerancia a fallos.

Se observa el siguiente grafico de la Figura 11 - Comparativa Spray con otros servidores web, que spray obtiene los mejores resultados de rendimiento dentro de los servidores web que corren bajo la JVM, este grafico compara el numero de respuesta que el servidor es capaz de responder por segundo con diferentes hardware, en el eje vertical se usa una maquina Amazon EC2 mientras que en el eje horizontal se usa un servidor dedicado (Procesador I7).

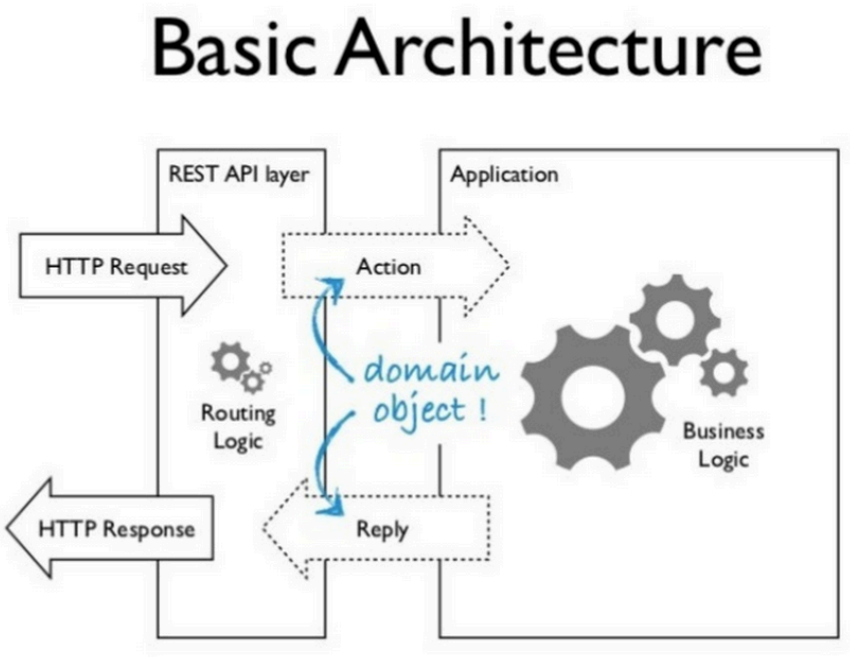


Figura 10 - Arquitectura REST

Figura 11 - Comparativa Spray con otros servidores web (Spray.io, 2013)

2.4.4. REST

Estilo de arquitectura de software para la construcción de servicios web escalables. REST (Representational State Transfer) es un estilo de arquitectura para desarrollar servicios. Los servicios web que siguen este estilo deben cumplir con las siguientes premisas.

Cliente/Servidor: Como servicios web son cliente servidor y definen una interfaz de comunicación entre ambos separando completamente las responsabilidades entre ambas partes. Para definir esta interfaz se establecen como principios los métodos HTTP GET, POST, PUT y DELETE, para realizar acciones de lectura, creación, actualización y eliminación respectivamente.

Sin estado: Son servicios web que no mantienen estado asociado al cliente. Esta capacidad se podría decir que viene intrínseca a la plataforma sobre la que se sustenta, HTTP. Cada petición que se realiza a ellos es completamente independiente de la siguiente . Todas las llamadas al mismo servicio serán idénticas.

Cache: El contenido de los servicios web REST ha se puede cachear de tal forma que una vez realizada la primera petición al servicio el resto puedan apoyarse en la cache si fuera necesario, como si de una pagina http se tratara.

No hubo en este caso comparación con otras opciones en el mercado ya que la aplicación del protocolo REST era parte de los requisitos no funcionales.

2.4.5. Servidor de bases de datos

La base de datos usada para desarrollar el proyecto es una base de datos relacional Opensource denominada Postgresql, como muchos otros proyectos de código abierto, no tiene el respaldo de una compañía, mas bien es mantenida por la comunidad que trabaja en ella de manera altruista y libre. (PostgreSQL)

Las características principales de Postgres son:

- Alta concurrencia

- Amplia variedad de tipos nativos

Siempre ha sido comparada con las diferentes y nuevas opciones en el mercado de bases de datos NoSql, debido al boom de estas ultimas, se pensó que en cierto punto, las bases de datos relacionales serian sustituidas. Al cabo de un tiempo se ha probado que desde el punto de vista del diseño o del rendimiento, las bases de datos no relacionales no siempre son útiles en determinados dominios.

Estudios han demostrado el rendimiento de Postgres comparado con MongoDb (EnterpriseDB, 2014), es superior en la mayoría de los casos.

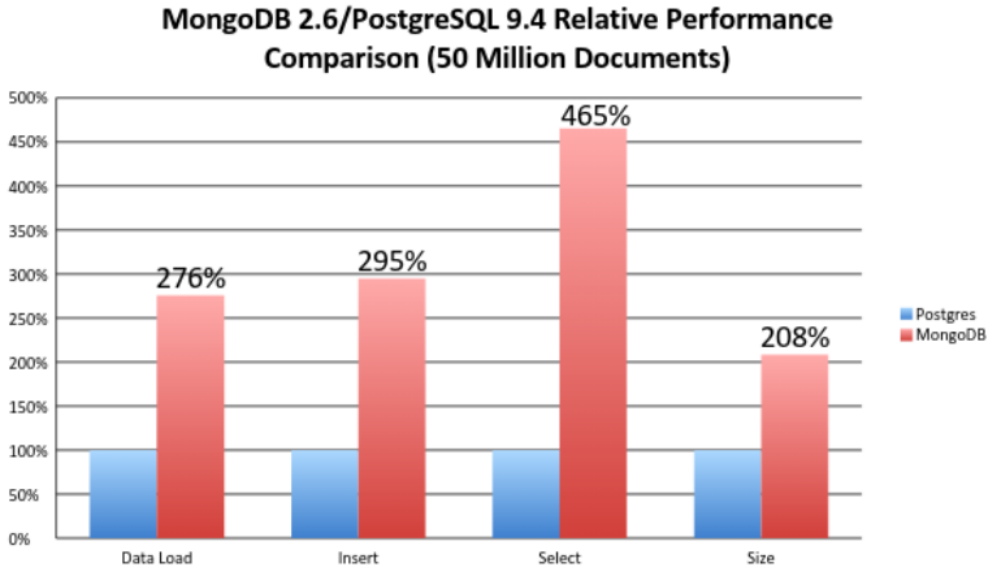


Figura 12 - Comparativa PostgreSQL vs MongoDB (EnterpriseDB, 2014)

Se observa como Postgres es claramente superior en operaciones de lectura, insertando 50 millones de registros en se observan además estos datos (Figura 13 - Datos comparativos Postgres vs MongoDB):

- La inserción de altas cantidades de datos fue aproximadamente el doble de rápido en Postgres.
- MongoDB consume aproximadamente un 33% mas de disco duro.
- La inserción de datos tarda tres veces mas en MongoDB.
- La lectura de datos es dos veces y medio mas lento en MongoDB.

	Postgres	MongoDB
Data Load (s)	4,732	13,046
Insert (s)	29,236	86,253
Select (s)	594	2,763
Size (GB)	69	145

Figura 13 - Datos comparativos Postgres vs MongoDB

Con estos datos se pretende hacer ver que no siempre lo último o lo que está a la moda puede ser siempre la mejor solución para los proyectos.

La posibilidad de desplegar una base de datos dentro del entorno de Heroku también contribuyó a tomar la decisión de usar esta base de datos relacional.

2.4.6. Herramienta de compilación y gestión de dependencias

La herramienta usada para la compilación es SBT, similar a Maven o Ant, SBT está enfocada al código Scala.

La manera de definir las opciones de configuración y dependencias es exactamente como si de código Scala se tratase. La estructura de los proyectos es compartida con el formato de Maven (Figura 14 - Estructura de proyecto SBT).

```
src/
  main/
    resources/
       <files to include in main jar here>
    scala/
       <main Scala sources>
    java/
       <main Java sources>
  test/
    resources
       <files to include in test jar here>
    scala/
       <test Scala sources>
    java/
       <test Java sources>
```

Figura 14 - Estructura de proyecto SBT

Sbt usa Apache Ivy para la gestión de dependencias, este a su vez es encargado de descargar los jar desde un repositorio central de Maven (Typesafe, 2014).

En el proyecto se usa SBT para la compilación y lanzamiento de tests.

2.4.7. Swagger

Swagger es una herramienta cuya finalidad es la generación automática de documentación para una API REST, usando anotaciones en el código, Swagger genera un cliente en el navegador que se comunica con el servidor mediante peticiones REST, el servidor envía los metadatos en formato JSON y estos son presentados de manera elegante y concisa al usuario a través del navegador (Figura 15 - Presentación de la API mediante Swagger).

La integración con una API siempre es una tarea ardua para un desarrollador, la falta de documentación y las distintas implementación del protocolo REST hacen que el mecanismo de desarrollo contra una API sea siempre prueba y error.

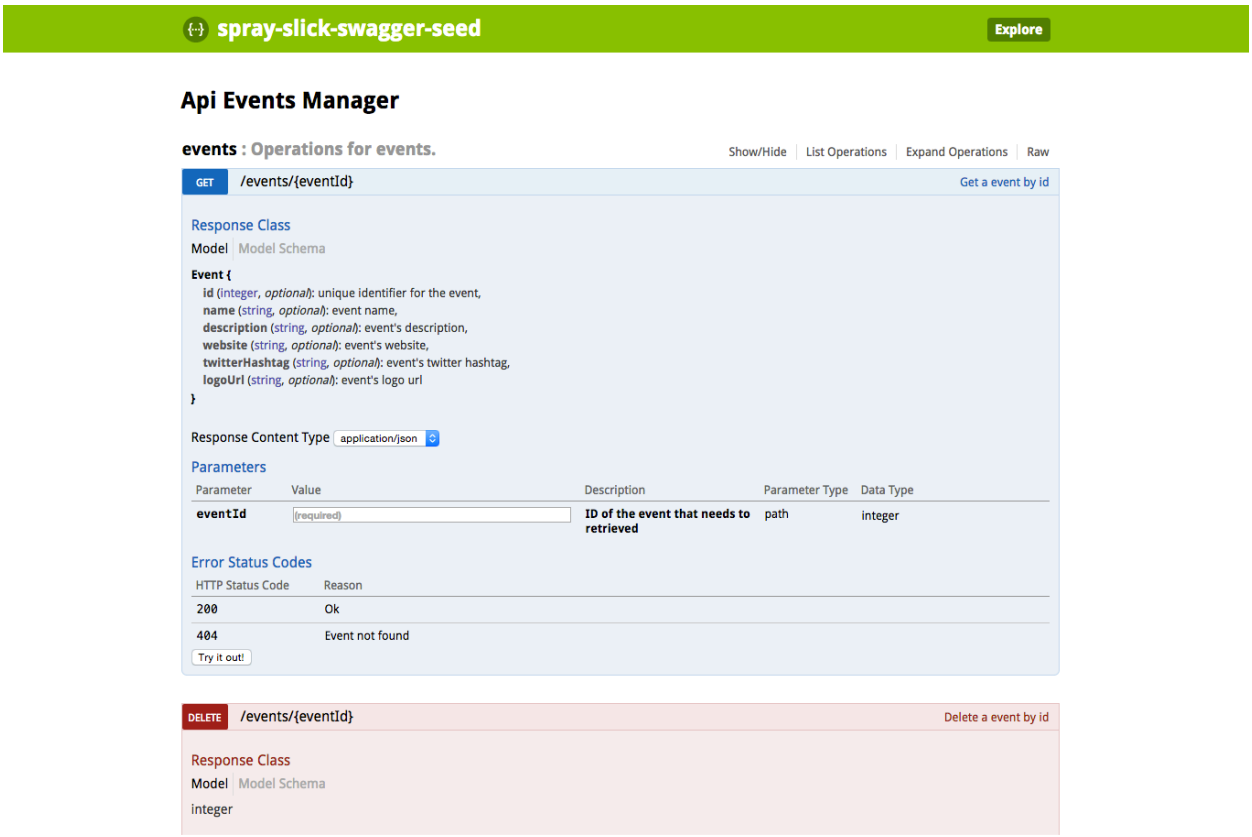


Figura 15 - Presentación de la API mediante Swagger

3. Herramientas para la elaboración del proyecto

3.1. Plantilla del proyecto (Seed en Activator)

El trabajo se ideó desde sus inicios como un proyecto open source, es decir se quería colaborar con la comunidad con un código de calidad que pueda ser reusado en diferentes proyectos incluso desde el ámbito comercial, para ello el proyecto se dividió en dos partes.

La primera parte sería una plantilla (seed) que se abstrajera de la lógica de negocio y sirviera para presentar a la comunidad, este proyecto formaría parte de los templates que ofrece Typesafe, (https://www.typesafe.com/activator/templates). La plantilla fue publicada con el nombre de

"Spray swagger slick3 authentication seed" en (Figura 16 - Plantilla en Typesafe) (https://www.typesafe.com/activator/template/spray-swagger-slick3-seed)

Este "*seed*" contiene la infraestructura básica del proyecto (dependencias, frameworks, etc.), en el momento de su presentación tuvo una buena acogida dentro de la comunidad como así lo demuestran los "stars" que tiene en GitHub (Figura 17 - Pantalla del código de la plantilla en).

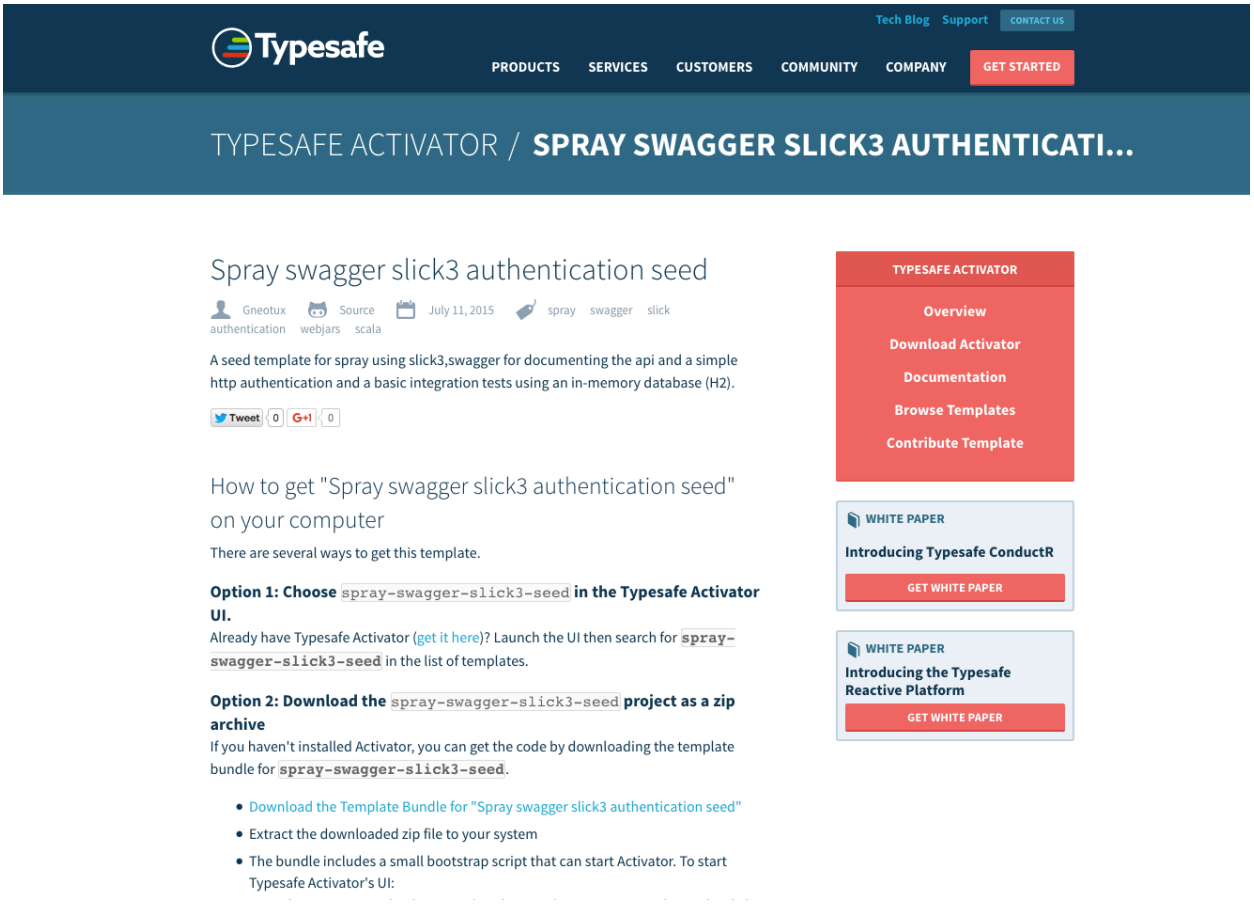


Figura 16 - Plantilla en Typesafe

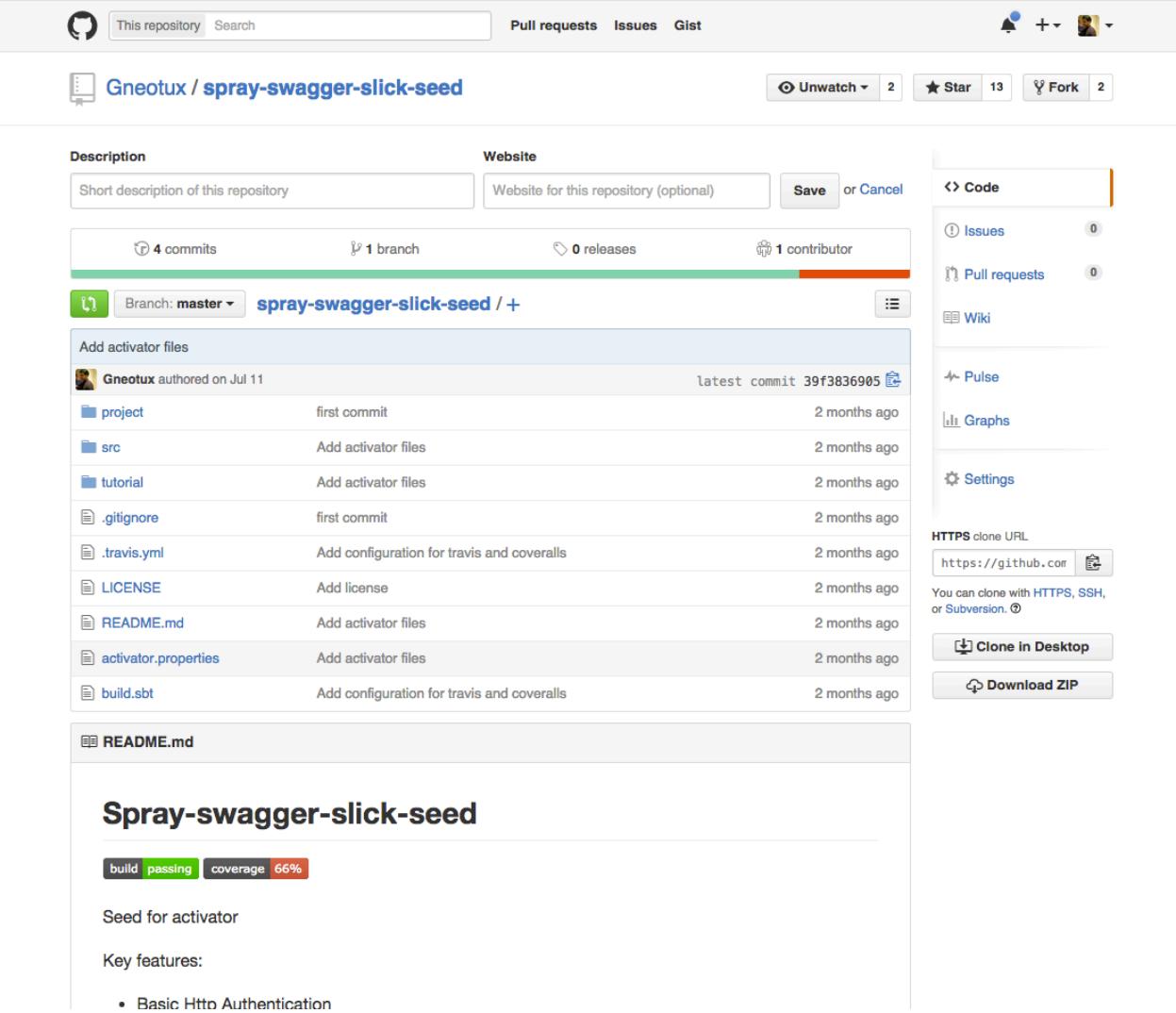


Figura 17 - Pantalla del código de la plantilla en GitHub

3.2. Tests y Cobertura del código

La cobertura del código mediante tests es una parte fundamental dentro de cualquier proyecto que se considere poner en producción, en este desarrollo se ha llevado a cabo este principio por lo que se ha realizado código de tests para casi la totalidad de los distintos recursos de la API. El énfasis por ofrecer esta plataforma como software libre ha incidido en la realización de una amplia cobertura de tests.

Existen diferentes herramientas que permiten conocer el porcentaje de código que esta cubierto por tests, en este proyecto se ha usado un plugin para SBT denominado Scoverage, una herramienta que realiza un chequeo del código cada vez que se corren los test y arroja un resultado.

En nuestro proyecto una vez finalizado se obtuvo una cobertura de tests del **76%**

3.3. Servidor de integración continua

La integración continua (continuous integration en inglés) es un modelo informático propuesto inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de pruebas de todo un proyecto.

El proceso suele ser: cada cierto tiempo (horas), descargarse las fuentes desde el control de versiones (por ejemplo CVS, Git, Subversion, Mercurial o Microsoft Visual SourceSafe) compilarlo, ejecutar pruebas y generar informes.

Para esto se utilizan aplicaciones como Solano CI, Bamboo, Continuum, Hudson, Jenkins, CruiseControl o Anthill (para proyectos Java) o CruiseControl.Net, Team Foundation Build para .Net, que se encargan de controlar las ejecuciones, apoyadas en otras herramientas como Ant o Maven (también para proyectos Java), o Nant o MSBUILD (para .Net) que se encargan de realizar las compilaciones, ejecutar las pruebas y realizar los informes.

A menudo la integración continua está asociada con las metodologías de programación extrema y desarrollo ágil. (Wikipedia, Integracion continua)

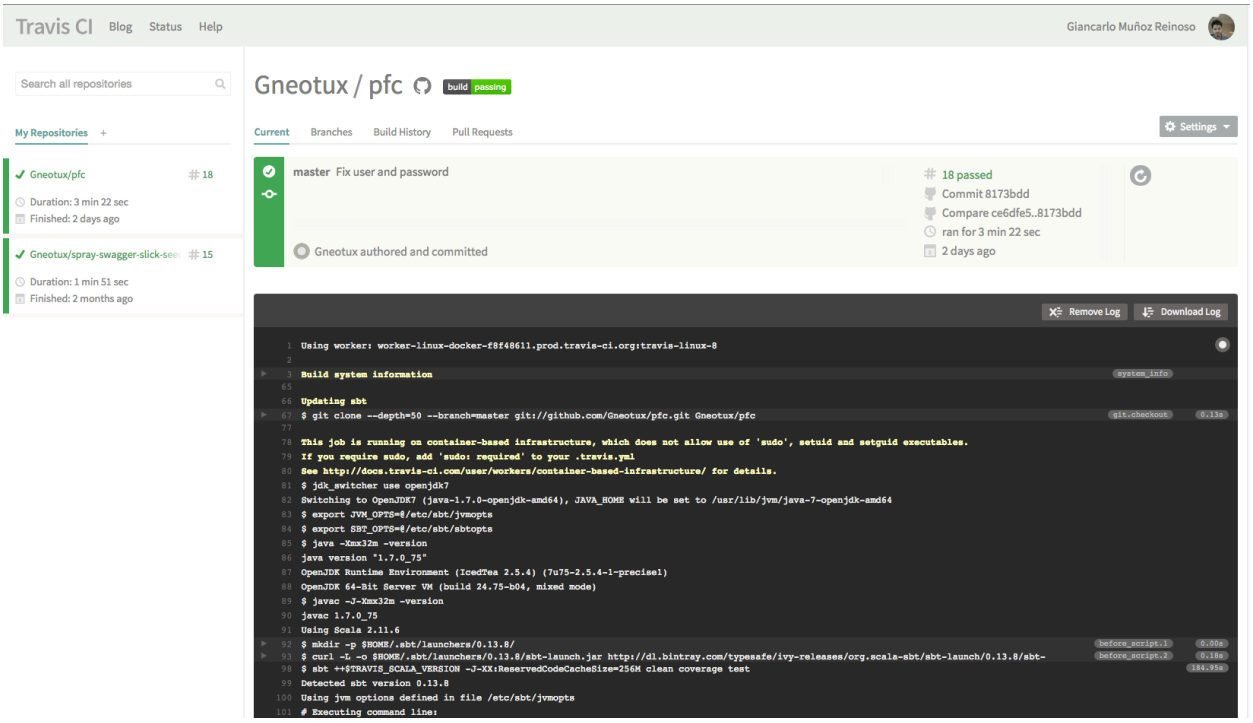


Figura 18 - Travis CI

En nuestro caso se ha usado una solución que se conecta automáticamente con nuestro repositorio GitHub y lanza los test cada vez que nuevo código es introducido en el.

Este producto se denomina Travis (https://travis-ci.org/), este se descarga una nueva versión del repositorio cada vez que detecta un cambio y se encarga de compilar y lanzar los test.

El site con la información de nuestro proyecto es: https://travis-ci.org/Gneotux/pfc

Además de esta herramienta de CI, se ha instalado en el repositorio el plugin Coveralls (https://coveralls.io/) , este plugin obtiene la cobertura del código obteniendo desde Travis los archivos que contienen la información al porcentaje de test.

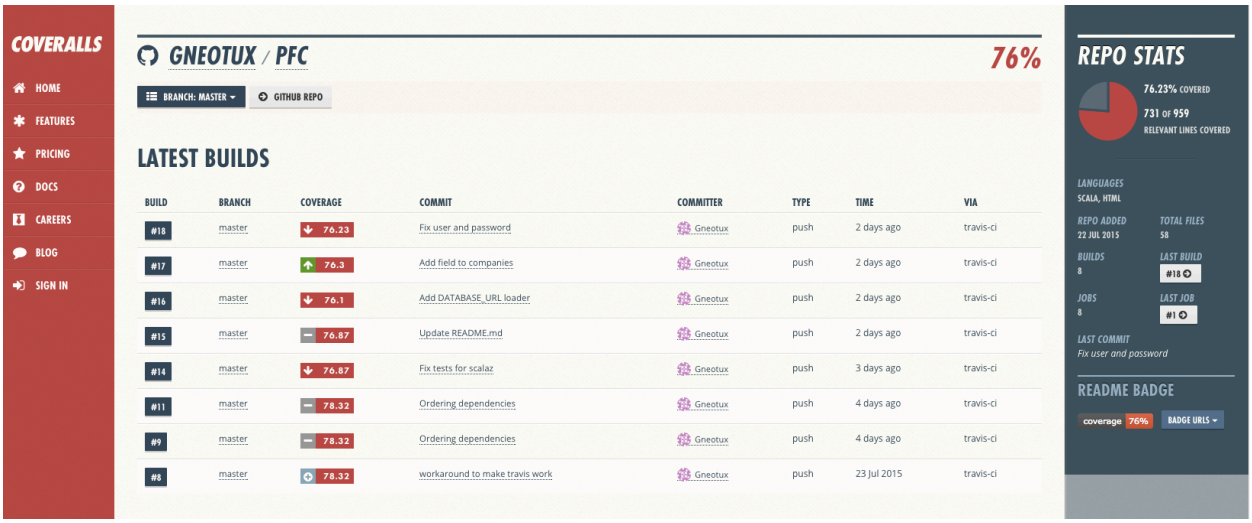


Figura 19 - Coveralls

3.4. Entorno de desarrollo

3.4.1. Intellij

IntelliJ Idea es el entorno integrado de desarrollo usado (Integrated development environment) durante el proyecto, originalmente ideado para Java. Intellij posee una serie de funcionalidades que la hacen muy atractivas desde el punto de vista del desarrollador, para empezar se podría decir que tiene plugins para casi todas las necesidades del desarrollador, uno de ellos es el plugin de Scala, que permite como si de código Java se tratara obtener los tipos, auto-completaciones, errores de compilación, etc. Otra brillante funcionalidad del programa es la integración completa con el sistema de control de versiones, en este caso Git. También proporciona una herramienta que permite lanzar los test individualmente obteniendo los resultados todo dentro de la misma pantalla.

Se decidió usar IntelliJ aun siendo de pago debido a la experiencia de este con el desarrollador en el ámbito profesional.

3.5. Entorno de trabajo

Para la realización de la aplicación se ha usado un Ordenador portátil Macbook Pro, procesador i7 2,3 ghz, 8gb Ram, 256 gb Disco duro ssd.

4. Desarrollo del proyecto

4.1. Fase Inicial

El proyecto fue desarrollado desde el ámbito de las metodologías Agiles, dentro de estas se uso Scrum, como metodología usando Jira como herramienta para llevar a cabo la creación de distintas tareas y la planificación de estas.

4.1.1. Scrum

Scrum es un proceso en el que se aplican de manera regular un conjunto de mejores prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto.

En Scrum se realizan entregas parciales y regulares del producto final, siendo estas entregas orquestadas por el denominado *Product Owner,* que es el encargado de priorizar las tareas. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es

aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto. (IBM, 2010)

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (en nuestro proyecto se fijó en 2 semanas). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite. Es decir el cliente siempre tiene constancia de que al final de cada iteración ha añadido valor a la empresa, se deja a un lado los proyectos "black-box" donde el cliente no sabe muy bien en qué estado se encuentra su producto y si este sigue los requisitos definidos a priori.

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente prioriza los objetivos balanceando el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas. De manera regular el cliente puede maximizar la utilidad de lo que se desarrolla y el retorno de inversión mediante la re planificación de objetivos que realiza al inicio de cada iteración.

4.1.2. Planificación de la iteración

Se va a explicar la manera de trabajo de forma general, es decir en el ámbito laboral, para después explicar consecuente adaptación que este proyecto requería, ya que no existe un equipo y el desarrollador es único.

El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:

1. Selección de requisitos (4 horas máximo). El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita. En nuestro caso la selección y priorización vino por parte del desarrollador.

2. Planificación de la iteración (4 horas máximo). El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos a que se ha comprometido. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se auto asignan las tareas. En este caso un único desarrollador era asignado a todas las tareas.

4.1.3. Herramienta de gestión de tareas

Para la gestión de nuestro Scrum se usó Jira. Jira es una herramienta para la planificación y seguimiento de un proyecto dentro del ámbito de trabajo de Scrum, provee la funcionalidad que un equipo completo necesita.

En la Figura 20 - Pantalla principal de Jira se observa la interfaz grafica que esta provee.

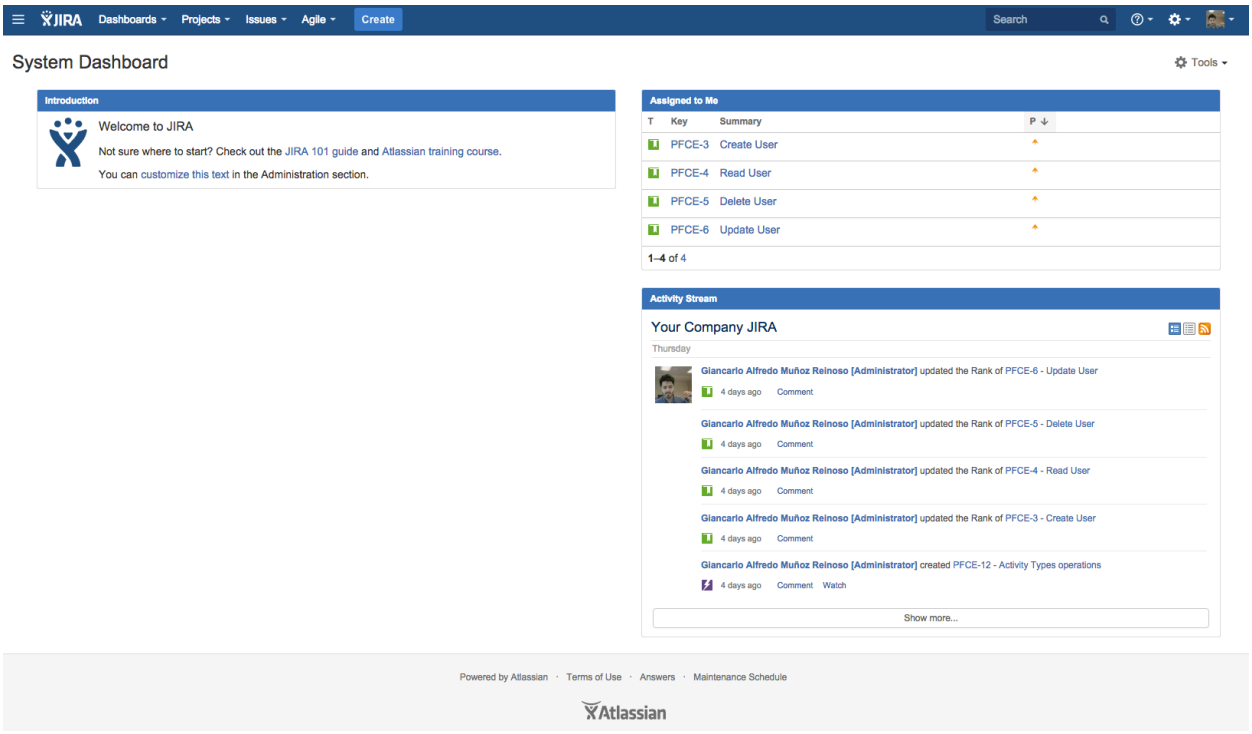


Figura 20 - Pantalla principal de Jira

4.1.4. Especificación de requisitos de usuarios

El requisito fundamental era la creación de una API para para la gestión de eventos orientado al "T3chFest".

"T3chFest"es un evento que se celebra anualmente con una duración determinada, normalmente 2 días, pudiendo sin embargo tener un numero arbitrario de días (normalmente consecutivos). En cada día el evento empieza a una determinada hora y finaliza a otra.

Los eventos están formados por actividades, la mayoría de las actividades son charlas o talleres de trabajo donde los asistentes colaboran junto con el ponente para realizar cierta tarea (pequeña pagina web, hola mundo de un lenguaje, etc.), cada una de estas actividades puede tener uno o mas

ponentes. Se posee la siguiente información de los ponentes: nombre, apellidos, información de contacto (email, twitter, linkedin, etc.), una pequeña introducción de si mismos, y pueden tener foto. También pueden estar promocionados por una compañía

Las actividades poseen un tema o tag, estos tags están formados por: nombre, nombre corto y color.

Además de estos campos las actividades tienen, titulo, descripción, hora de inicio, hora de fin, localización, hashtag (Twitter) y una serie de recursos, estos recursos pueden estar formados por diferentes productos (archivo comprimido con código, links, pdf con instrucciones, powerpoint con diapositivas, un enlace a *Youtube*, etc.)

Con estos requisitos se paso a crear las distintas tareas que iban a ser introducidas en la lista denominada "backlog" en Jira.

En las siguientes tablas se observan las tareas definidas para el proyecto, estas tareas forman parte de distintos Hitos (Epic), estos hitos son definidos para señalar una parte de la funcionalidad de la aplicación. Los hitos son los siguientes:

[PFCE-3] Create, Read, Update and Delete operations for users.	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Epic	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	14		
Original Estimate:	16		

Epic Name:	User API
Issues in Epic:	**[PFCE-12] Create User, [PFCE-13] Delete User, [PFCE-14] Update User, [PFCE-15] Get User**

[PFCE-4] Create, Read, Update and Delete operations for activities.	
Status:	Done
Project:	Proyecto Fin Carrera-Gestation Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Epic	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	25		
Original Estimate:	48		

Epic Name:	Activity API
Issues in Epic:	**[PFCE-16] Create Activity, [PFCE-17] Delete Activity, [PFCE-18] Update Activity, [PFCE-19] Get Activity, [PFCE-44] Add Attendee to Activity, [PFCE-45] Delete Attendee from Activity, [PFCE-46] Get all attendees in Activity, [PFCE-47] Add Speaker to Activity, [PFCE-48] Delete Speaker from Activity, [PFCE-49] Get all Speakers in Activity**

[PFCE-5] Create, Read, Update and Delete operations for events.	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Epic	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	21		
Original Estimate:	31		

Epic Name:	Event API
Issues in Epic:	**[PFCE-20] Create Event, [PFCE-21] Delete Event, [PFCE-22] Update Event, [PFCE-23] Get Event, [PFCE-50] Add Sponsor to Event , [PFCE-51] Delete Sponsor from Event, [PFCE-52] Get all sponsors in Event**

[PFCE-6] Create, Read, Update and Delete operations for locations.	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Epic	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	10		
Original Estimate:	15		

Epic Name:	Location API
Issues in Epic:	**[PFCE-24] Create Location, [PFCE-25] Delete Location, [PFCE-26] Update Location, [PFCE-27] Get Location**

[PFCE-7] Create, Read, Update and Delete operations for companies.	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Epic	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	11		
Original Estimate:	12		

Epic Name:	Company API
Issues in Epic:	**[PFCE-28] Create Company, [PFCE-29] Delete Company, [PFCE-30] Update Company, [PFCE-31] Get Company**

[PFCE-8] Create, Read, Update and Delete operations for Event Days.	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Epic	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	10		
Original Estimate:	12		

Epic Name:	EventDay API
Issues in Epic:	**[PFCE-32] Create EventDay, [PFCE-33] Delete EventDay, [PFCE-34] Update EventDay, [PFCE-35] Get EventDay**

[PFCE-9] Create, Read, Update and Delete operations for tags.	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Epic	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	9		
Original Estimate:	9		

Epic Name:	Tag API
Issues in Epic:	**[PFCE-36] Create Tag, [PFCE-37] Delete Tag, [PFCE-38] Update Tag, [PFCE-39] Get Tag**

[PFCE-10] Create, Read, Update and Delete operations for activity types.	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Epic	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	6		
Original Estimate:	8		

Epic Name:	ActivityType API
Issues in Epic:	**[PFCE-40] Create ActivityType, [PFCE-41] Delete ActivityType, [PFCE-42] Update ActivityType, [PFCE-43] Get ActivityType**

[PFCE-11] Deploy the application	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Epic	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	21		
Original Estimate:	29		

Epic Name:	Deploy the Application
Issues in Epic:	**[PFCE-53] Add continuous integration system, [PFCE-54] Create Database Script, [PFCE-55] Set-up Heroku**

[PFCE-12] Create User	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	5		
Original Estimate:	4		

Epic Link:	User API
Sprint:	Sprint 1

[PFCE-13] Delete User	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	4		
Original Estimate:	5		

Epic Link:	User API
Sprint:	Sprint 1

[PFCE-14] Update User	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	4		
Original Estimate:	4		

Epic Link:	User API
Sprint:	Sprint 1

[PFCE-15] Get User	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	3		

Epic Link:	User API
Sprint:	Sprint 1

[PFCE-16] Create Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	4		

Epic Link:	Activity API
Sprint:	Sprint 1

[PFCE-17] Delete Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	5		

Epic Link:	Activity API
Sprint:	Sprint 1

[PFCE-18] Update Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	6		
Original Estimate:	4		

Epic Link:	Activity API
Sprint:	Sprint 1

[PFCE-19] Get Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	4		

Epic Link:	Activity API
Sprint:	Sprint 1

[PFCE-20] Create Event	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	3		

Epic Link:	Event API
Sprint:	Sprint 1

[PFCE-21] Delete Event	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	3		

Epic Link:	Event API
Sprint:	Sprint 1

[PFCE-22] Update Event	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	6		
Original Estimate:	3		

Epic Link:	Event API
Sprint:	Sprint 1

[PFCE-23] Get Event	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	3		

Epic Link:	Event API
Sprint:	Sprint 1

[PFCE-24] Create Location	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	3		

Epic Link:	Location API
Sprint:	Sprint 1

[PFCE-25] Delete Location	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	3		

Epic Link:	Location API
Sprint:	Sprint 1

[PFCE-26] Update Location	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	6		

Epic Link:	Location API
Sprint:	Sprint 1

[PFCE-27] Get Location	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	3		

Epic Link:	Location API
Sprint:	Sprint 1

[PFCE-28] Create Company	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	2		

Epic Link:	Company API
Sprint:	Sprint 1

[PFCE-29] Delete Company	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	2		

Epic Link:	Company API
Sprint:	Sprint 1

[PFCE-30] Update Company	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	4		
Original Estimate:	3		

Epic Link:	Company API
Sprint:	Sprint 1

[PFCE-31] Get Company	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	3		

Epic Link:	Company API
Sprint:	Sprint 1

[PFCE-32] Create EventDay	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	3		

Epic Link:	EventDay API
Sprint:	Sprint 1

[PFCE-33] Delete EventDay	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	3		

Epic Link:	User API
Sprint:	Sprint 1

[PFCE-34] Update EventDay	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	6		
Original Estimate:	3		

Epic Link:	EventDay API
Sprint:	Sprint 1

[PFCE-35] Get EventDay	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	3		

Epic Link:	User API
Sprint:	Sprint 1

[PFCE-36] Create Tag	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	3		

Epic Link:	Tag API
Sprint:	Sprint 1

[PFCE-37] Delete Tag	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	2		

Epic Link:	Tag API
Sprint:	Sprint 1

[PFCE-38] Update Tag	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	2		

Epic Link:	Tag API
Sprint:	Sprint 1

[PFCE-39] Get Tag	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	2		

Epic Link:	Tag API
Sprint:	Sprint 1

[PFCE-40] Create ActivityType	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	2		

Epic Link:	ActivityType API
Sprint:	Sprint 1

[PFCE-41] Delete ActivityType	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	2		

Epic Link:	ActivityType API
Sprint:	Sprint 1

[PFCE-42] Update ActivityType	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	2		

Epic Link:	ActivityType API
Sprint:	Sprint 1

[PFCE-43] Get ActivityType	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	2		

Epic Link:	ActivityType API
Sprint:	Sprint 1

[PFCE-44] Add Attendee to Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	3		
Original Estimate:	6		

Epic Link:	Activity API
Sprint:	Sprint 1

[PFCE-45] Delete Attendee from Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	4		
Original Estimate:	6		

Epic Link:	Activity API
Sprint:	Sprint 1

[PFCE-46] Get all attendees in Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	4		

Epic Link:	Activity API
Sprint:	Sprint 1

[PFCE-47] Add Speaker to Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	1		
Original Estimate:	6		

Epic Link:	User API
Sprint:	Sprint 1

[PFCE-48] Delete Speaker from Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	1		
Original Estimate:	6		

Epic Link:	User API
Sprint:	Sprint 1

[PFCE-49] Get all Speakers in Activity	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	1		
Original Estimate:	4		

Epic Link:	User API
Sprint:	Sprint 1

[PFCE-50] Add Sponsor to Event	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	4		
Original Estimate:	6		

Epic Link:	Event API
Sprint:	Sprint 1

[PFCE-51] Delete Sponsor from Event	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	6		

Epic Link:	Event API
Sprint:	Sprint 1

[PFCE-52] Get all sponsors in Event	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	2		
Original Estimate:	4		

Epic Link:	Event API
Sprint:	Sprint 1

[PFCE-53] Add continuous integration system	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	8		
Original Estimate:	12		

Epic Link:	Deploy the application
Sprint:	Sprint 1

[PFCE-54] Create Database Script	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	6		
Original Estimate:	5		

Epic Link:	Deploy the application
Sprint:	Sprint 1

[PFCE-55] Set-up Heroku	
Status:	Done
Project:	Proyecto Fin Carrera-Gestion Eventos
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Giancarlo Alfredo Muñoz Reinoso [Administrator]	**Assignee:**	Giancarlo Alfredo Muñoz Reinoso [Administrator]
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0		
Time Spent:	7		
Original Estimate:	12		

Epic Link:	Deploy the application
Sprint:	Sprint 1

Para la estimación se ha usado un diagrama Gantt en el que se ha contabilizado la jornada laboral como un periodo de 7 horas en el que se ha contabilizado la jornada laboral como un periodo de 7 horas. Se han contabilizado los hitos como unidad de trabajo.

Proyecto Fin de Carrera, Gestion de eventos					
Section 1 - Analisis	**06/03/15**	**06/08/15**		**100%**	**4 d**
Definir los requisitos de usuario	06/03/15	06/08/15	Alejandro Baldominos	100%	4 d
Section 2 - Desarrollo	**06/09/15**	**07/13/15**		**100%**	**25 d**
[PFCE-3] Create, Read, Update and Delete operations for users.	06/09/15	06/11/15	Giancarlo Munoz	100%	3 d
[PFCE-4] Create, Read, Update and Delete operations for activities.	06/12/15	06/22/15	Giancarlo Munoz	100%	7 d
[PFCE-5] Create, Read, Update and Delete operations for events.	06/23/15	06/29/15	Giancarlo Munoz	100%	5 d
[PFCE-6] Create, Read, Update and Delete operations for locations.	06/30/15	07/01/15	Giancarlo Munoz	100%	2d
[PFCE-7] Create, Read, Update and Delete operations for companies.	07/02/15	07/03/15	Giancarlo Munoz	100%	2d
[PFCE-8] Create, Read, Update and Delete operations for eventDays.	07/06/15	07/07/15	Giancarlo Munoz	100%	2d
[PFCE-9] Create, Read, Update and Delete operations for tags.	07/08/15	07/09/15	Giancarlo Munoz	100%	2d
[PFCE-10] Create, Read, Update and Delete operations for activityTypes.	07/10/15	07/13/15	Giancarlo Munoz	100%	2d
Section 3 - Despliegue	**07/14/15**	**07/17/15**		**100%**	**5 d**
[PFCE-11] Deploy the application	07/14/15	07/17/15	Giancarlo Munoz	100%	5 d

Figura 21 - Diagrama Gantt parte 1

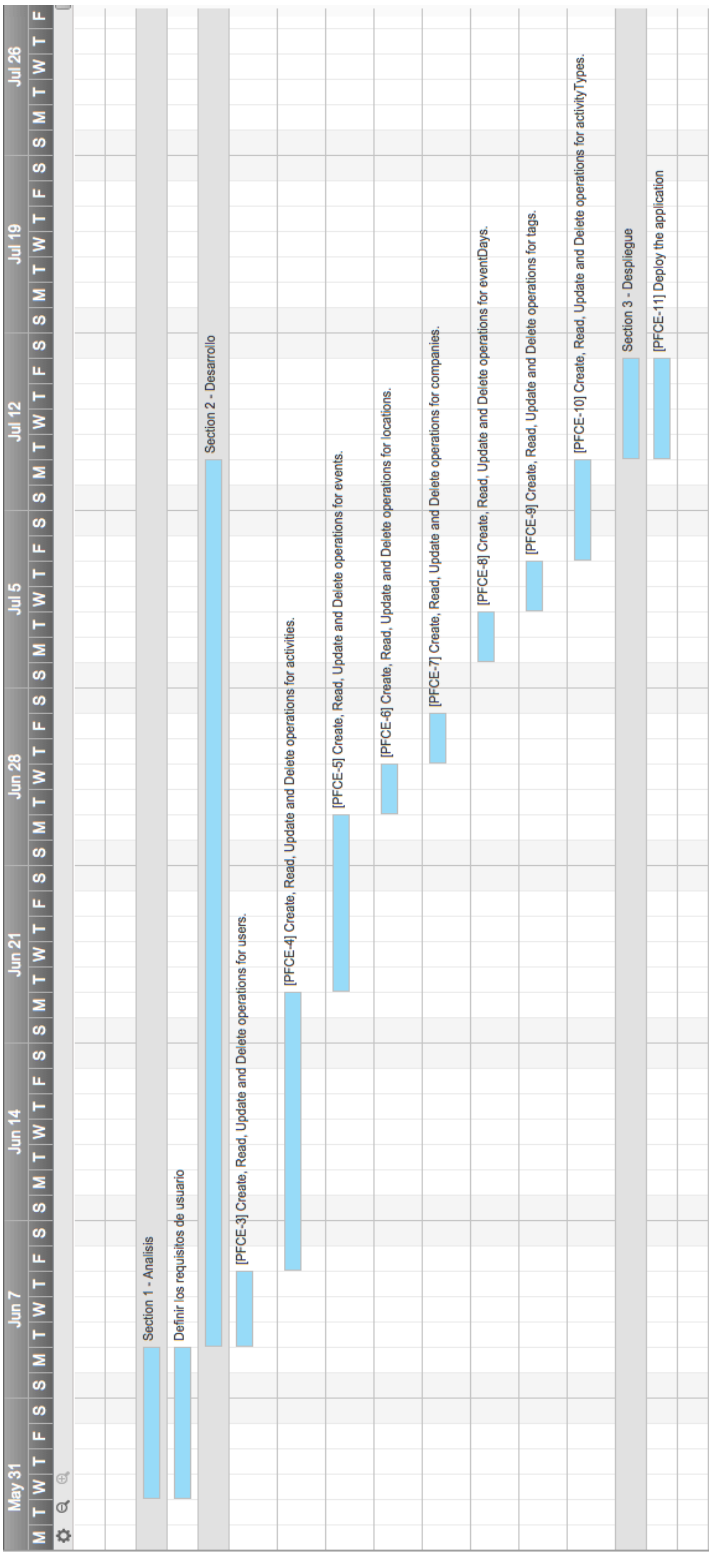


Figura 22 - Diagrama Gantt parte 2

4.1.5. Presupuesto

Para la realización de este proyecto se han considerado los siguientes recursos todos los costes se presuponen para perfiles de "contract" profesionales que trabajan como autónomos para la realización de distintas tareas, el salario se establece por día de trabajo, 7 horas, no es necesario proveer a estos profesionales del equipo necesario únicamente si se desea usar servidores de pago o propios en los que habría que incluir los costes en el presupuesto. En este proyecto únicamente se han usado herramientas de software libre.

Analista: Profesional titulado en Ingeniería Informática. Encargado de la toma de requisitos y de definir las tareas en Jira. Los honorarios de este profesional se sitúan en 700 euros al día.

Programador: Profesional cualificado con altos conocimientos en programación funcional. Los honorarios de este profesional se sitúan en 600 euros al día.

Contabilizando los "Epics" según las horas estimadas tendríamos:

Epic	Horas estimadas
[PFCE-3] Create, Read, Update and Delete operations for users.	16
[PFCE-4] Create, Read, Update and Delete operations for activities.	48
[PFCE-5] Create, Read, Update and Delete operations for events.	31

[PFCE-6] Create, Read, Update and Delete operations for locations.	15
[PFCE-7] Create, Read, Update and Delete operations for companies.	12
[PFCE-8] Create, Read, Update and Delete operations for eventDays.	12
[PFCE-9] Create, Read, Update and Delete operations for tags.	9
[PFCE-10] Create, Read, Update and Delete operations for activityTypes.	8
[PFCE-11] Deploy the application	29
TOTAL	180

Los días de trabajo del analista se estiman en 4 días x 700 euros/día = 2800 euros

180 horas estimadas / 7 Horas = 25,7 Días ~> 26 dias x 600 euros/dia = 15600 euros

El presupuesto final se sitúa en **18400** EUROS.

4.2. Fase de desarrollo

4.2.1. Modelo de base de datos

El modelo de base de se muestra en la Figura 23 - Diagrama de base de datos

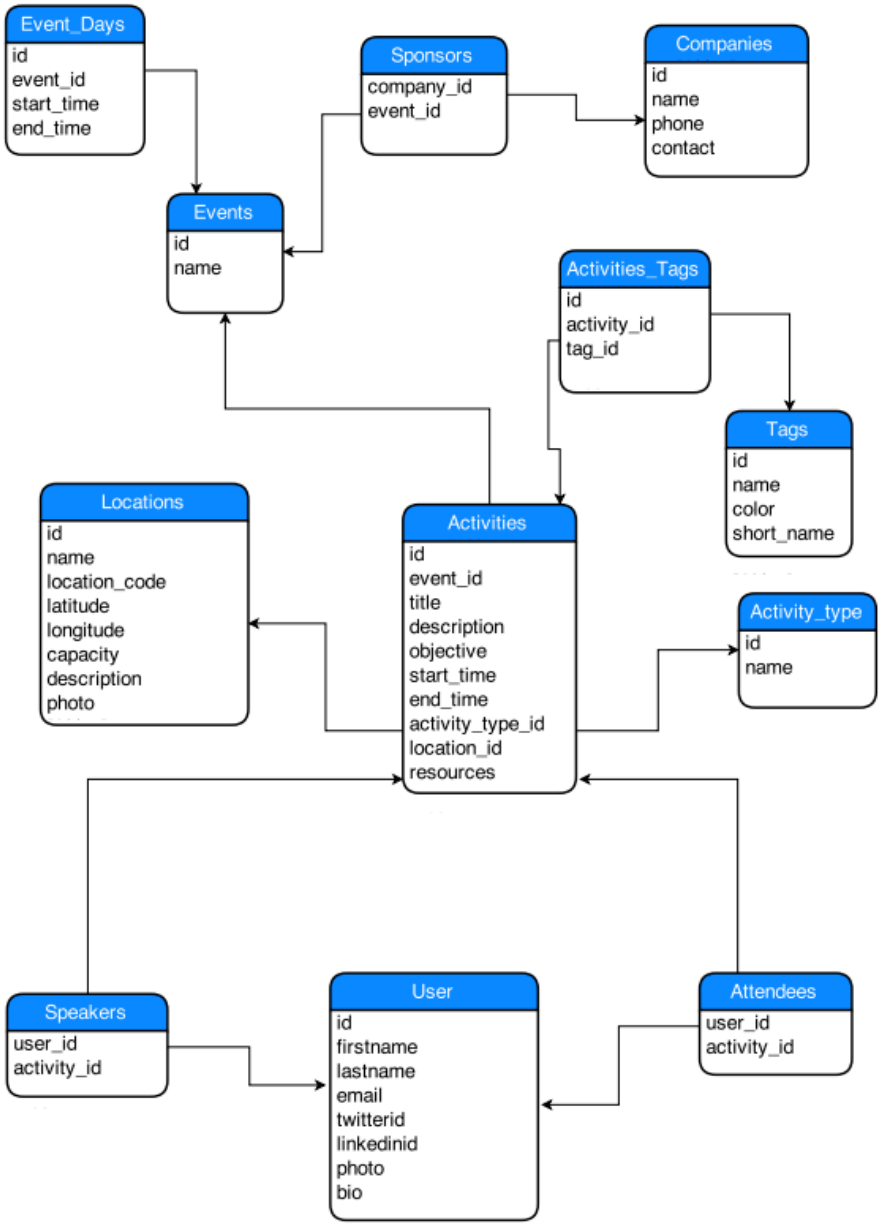


Figura 23 - Diagrama de base de datos

4.2.2. Diseño de la aplicación

La aplicación ha sido diseñada siguiendo buenas practicas dentro de la computación como es el principio de única responsabilidad, cada componente debe encargarse de una única función.

Existen cuatro capas en la aplicación, la capa denominada router, encargada de recibir las peticiones desde el cliente, la capa de servicio encargada de procesar las peticiones y realizar las conversiones entre los objetos de I/O y los objetos de negocio, la capa Dao desde la que se realizan las operaciones de acceso a la base de datos y el modelo que se encarga de representar las entidades de negocio.

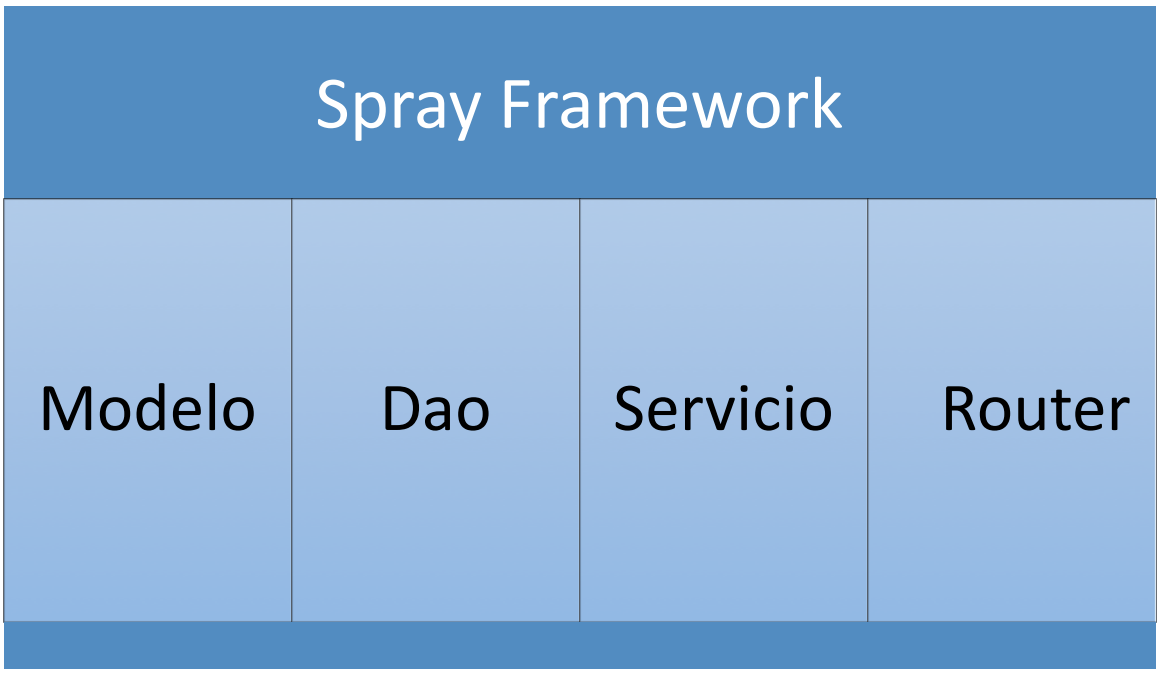


Figura 24 - Representación del diseño

4.2.3. Consideraciones sobre el desarrollo

El desarrollo ha seguido un proceso como si de un proyecto del ámbito profesional se tratara. Aun solo contando con un desarrollador, se han seguido las metodologías aplicadas a equipos de desarrollo de tamaño medio/grande.

Se muestra en las tareas además de la estimación, cuanto se tardo en realizar la tarea por lo que se podría hacer una comparación para futuros proyectos que permitirían realizar un mejor ajuste presupuestario del producto.

4.3. Resumen del proyecto

La oferta de API actualmente es casi obligatoria para cualquier servicio en la red, desde Twitter, pasando por Google Maps, Renfe, servicios de previsión meteorológica, personajes de marvel, etc.

La correcta implementación de una api sin embargo no es obligatoria por esto supone muchos quebraderos de cabeza cuando desarrolladores tienen que desarrollar contra un servicio. El protocolo REST ha ayudado bastante a lograr armonizar la comunicación entre aplicaciones pero aun no es suficiente.

Este proyecto quiere servir no solo como solución para el proyecto de crear una API para la Techfest, también se pretende servir como acelerador para otros proyectos en los que se quieran usar las ultimas tecnologías dentro del ámbito de la programación funcional y escalabilidad

5. Conclusiones

5.1. Futuras líneas de desarrollo

Al igual que la plantilla desarrollada como base para la aplicación, este desarrollo tiene mucho potencial para posibles mejoras y adaptaciones a las necesidades del cliente, la buena base y el uso de las ultimas tecnologías hace que el framework sea fácil de mantener incluso en un periodo de tres o cuatro años.

La incorporación de herramientas de testeo e integración continua facilitan bastante cualquier modificación del código.

Desde el punto de vista de nueva funcionalidad, se podría desarrollar el sistema de autentificación usando un protocolo Oauth2[1].

Ampliar la cobertura de tests así como la inclusión de test para el website de la documentación estaría fácilmente justificado.

[1] Oauth2 : Permite que aplicaciones obtengan autorizaciones para operar con recursos de un usuario en determinada plataforma.

6. Referencias

Beringer, F. (s.f.). *Fred Beringer*. Obtenido de http://www.fredberinger.com/2010/06/musings-on-the-future-of-data-and-predictive-analytics-3/

COLOANDCLOUD. (2015). *Heroku*. Obtenido de Heroku Overview: http://www.coloandcloud.com/heroku/

Daily, S. (2013). *Science Daily*. Obtenido de Big Data, for better or worse: 90% of world's data generated over last two years: http://www.sciencedaily.com/releases/2013/05/130522085217.htm

EnterpriseDB. (Septiembre de 2014). Obtenido de Postgres Outperforms MongoDB and Ushers in New Developer Reality: http://www.enterprisedb.com/postgres-plus-edb-blog/marc-linster/postgres-outperforms-mongodb-and-ushers-new-developer-reality

Git-SCM. (s.f.). *Git-scm*. Obtenido de Acerca del control de versiones: https://git-scm.com/book/es-ni/v1/Iniciando-Acerca-del-Control-de-Versiones

Gonzalez, A. (s.f.). *Base64: la codificacion mas util en criptografia.* Obtenido de http://albertx.mx/blog/base64/

Heroku. (s.f.). *Dynos and the Dyno Manager*. Obtenido de https://devcenter.heroku.com/articles/dynos

IBM. (2010). *IBM developer works*. Obtenido de Scrum como metodologia: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Rational+Team+Concert+for+Scrum+Projects/page/SCRUM+como+metodolog%C3%ADa

Ibm. (s.f.). *Scrum como metodologia*. Obtenido de Ibm developer Works: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Rational+Team+Concert+for+Scrum+Projects/page/SCRUM+como+metodolog%C3%ADa

Muñoz Reinoso, G. (July de 2015). *GitHub*. Obtenido de PFC: https://github.com/Gneotux/pfc

Muñoz Reinoso, G. (July de 2015). *Reddit*. Obtenido de Reddit/Scala: https://www.reddit.com/r/scala/comments/3d006k/i_just_published_my_very_first_template_in/

Muñoz Reinoso, G. (2015). *SpraySwaggerSlick3Seed*. Obtenido de TypeSafe: https://www.typesafe.com/activator/template/spray-swagger-slick3-seed

Ohara, D. (2009). *Green Data Center Blog*. Obtenido de Aws economics: http://greenm3.typepad.com/green_m3_blog/2009/12/amazon-web-services-economics-center-comparing-awscloud-computing-vs-co-location-vs-owned-data-center.html

PostgreSQL. (s.f.). *PostgreSQL*. Obtenido de http://www.postgresql.org/

Remde, K. (2011). *Full of I.T.* Obtenido de Saas,Paas and Iaas: http://blogs.technet.com/b/kevinremde/archive/2011/04/03/saas-paas-and-iaas-oh-my-quot-cloudy-april-quot-part-3.aspx

Rouse, M. (January de 2015). *SearchCloudComputing*. Obtenido de http://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS

SalesForce. (2014). *Paas*. Obtenido de https://www.salesforce.com/paas/overview/

Solutions, X. (2013). *How to Pick the Right Cloud Provider for Your Business* . Obtenido de xyfon.com: http://xyfon.com/how-pick-right-cloud-provider-your-business

Spray.io. (2013). *Benchmarking Spray*. Obtenido de Spray: http://spray.io/blog/2013-05-24-benchmarking-spray/

Typesafe. (2014). *scala-sbt*. Obtenido de Sbt Reference: http://www.scala-sbt.org/0.13/docs/index.html

TypeSafe. (s.f.). *Scala-Sbt Reference*. Obtenido de http://www.scala-sbt.org/0.13/docs/index.html

W3.org. (s.f.). *Http 1.1*. Obtenido de http://www.w3.org/Protocols/rfc2616/rfc2616.html

Wikipedia. (s.f.). *Computacion en la nube*. Obtenido de https://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube

Wikipedia. (s.f.). *Control de versiones*. Obtenido de https://es.wikipedia.org/wiki/Control_de_versiones

Wikipedia. (s.f.). *Integracion continua*. Obtenido de Integracion continua: https://es.wikipedia.org/wiki/Integraci%C3%B3n_continua

Ingeniería Técnica Informática de Gestión

PROYECTO FIN DE CARRERA

ANEXO I : MANUAL DEL DESARROLLADOR (ESTUDIO DEL STACK TECNOLÓGICO PARA EL DESARROLLO DE UN FRAMEWORK ESCALABLE ORIENTADO A APIS REST)

Autor: Giancarlo Alfredo Muñoz Reinoso
Tutor: Alejandro Baldominos Gómez

Leganés, Septiembre 2015

Tabla de contenidos

Índice de figuras

1. Introducción

Con este manual se pretende facilitar al desarrollador el mantenimiento de la aplicación gestora de eventos. Debido al énfasis en la realización de un código altamente legible, sin repeticiones y altamente testeado, el desarrollo de cualquier mantenimiento evolutivo o corrección de fallos es bastante sencillo e intuitivo.

Es recomendable tener cierta noción de alguna metodología de desarrollo orientado a test, como puede ser TDD (*Test Driven Development*) o BDD (*Behaviour Driven Development*), estas metodologías ayudan a producir un código de calidad y con lo que facilitará futuros mantenimientos además de proporcionar "documentación" que ayuda a entender la finalidad y funcionalidad del código fuente. Además el desarrollador debe de estar familiarizado con la sintaxis de Scala y la programación funcional.

Para empezar a desarrollar necesitaremos descargar el código fuente de la aplicación.

Existen dos opciones podemos hacer un fork del proyecto de GitHub

```
$ git clone https://github.com/Gneotux/pfc.git
```

O podemos descargar el proyecto desde el propio repositorio en Github:

(https://github.com/Gneotux/pfc/archive/master.zip)

Para lanzar los tests simplemente vamos a la carpeta donde esta el código y escribimos:

```
$ sbt test
```

Para arrancar la aplicación:

```
$ sbt run
```

Una vez obtenido el código procedemos a explicar cómo está estructurado y realizaremos un pequeño tutorial añadiendo un *endpoint* o *resource* a la aplicación.

Para la realización de este manual se ha usado el IDE IntelliJ IDEA.

2. Código fuente

El código está dentro de la carpeta *Scala* y separado en paquetes que representan las diferentes capas cada una de ellas con una única responsabilidad (Figura 1 - Código fuente completo).

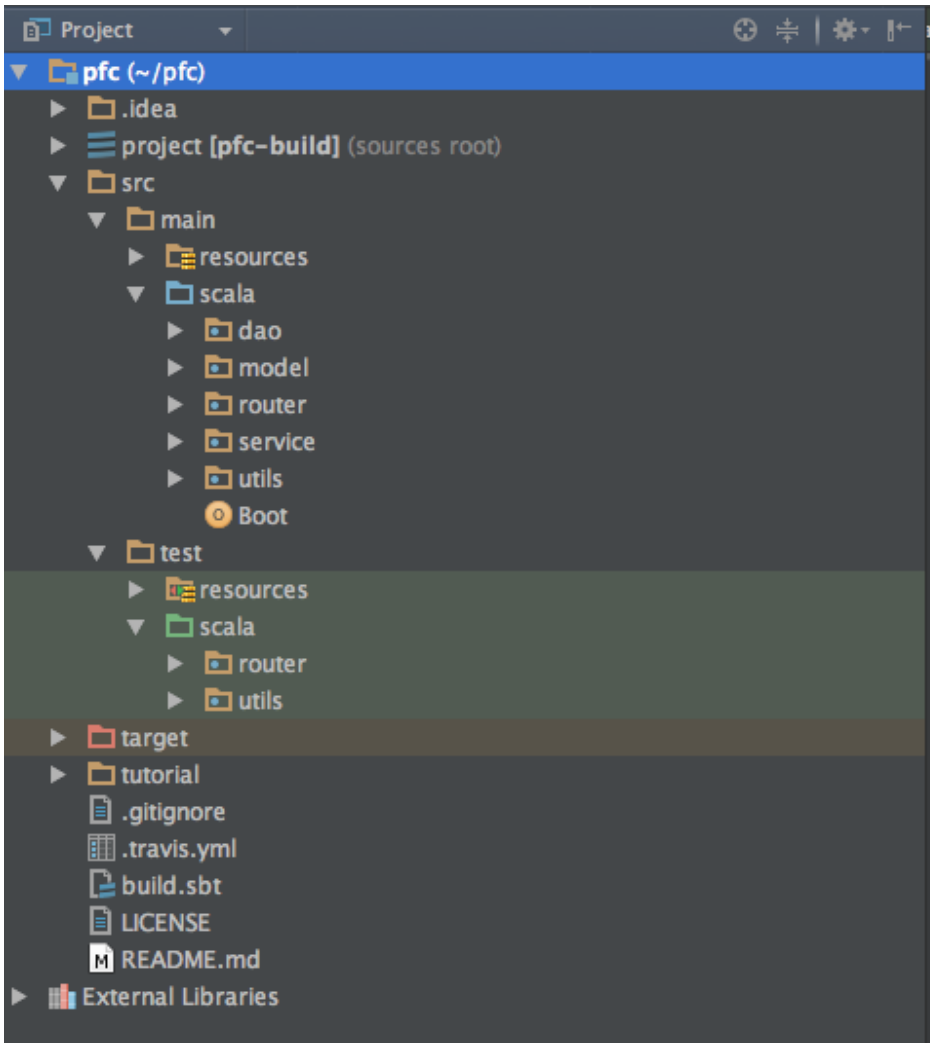


Figura 1 - Código fuente completo

2.1. Paquete *DAO*

El paquete dao (*Data Access Object*) es el encargado del acceso a la base de datos y provee una api que debe ser el único punto de entrada a los recursos almacenados en ella (Figura 2 - Paquete Dao).

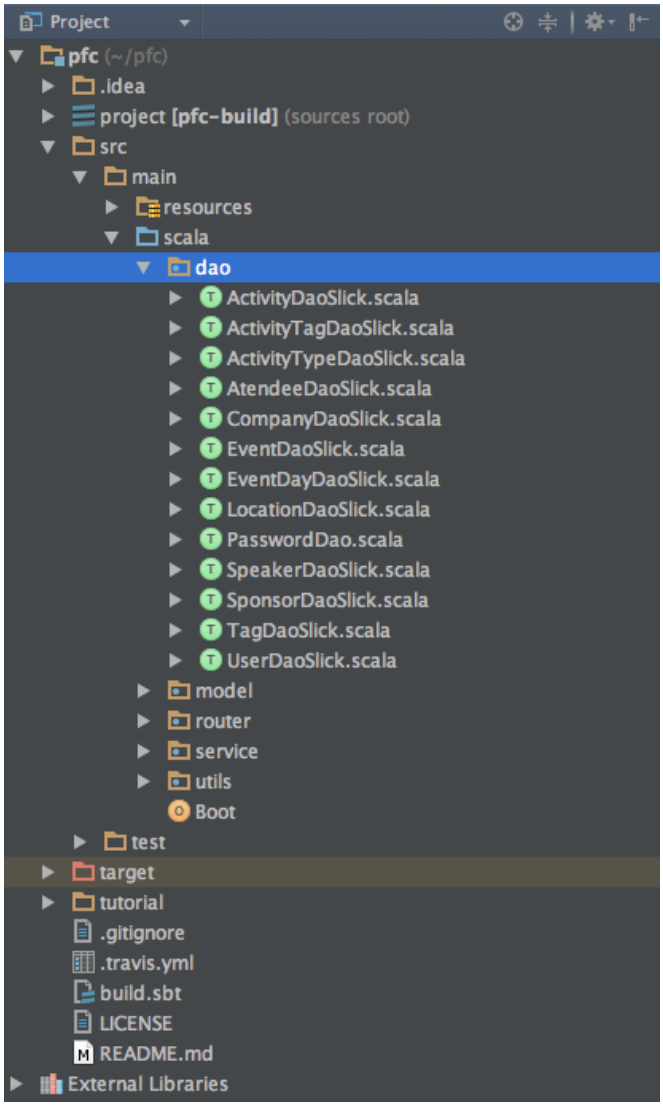


Figura 2 - Paquete Dao

2.2. Paquete *model*

El paquete *model* (Figura 3 - Paquete Model) contiene las clases que representan el modelo, es decir las entidades existentes en la aplicación tales como *User*, *Event*, *Activity*, etc.

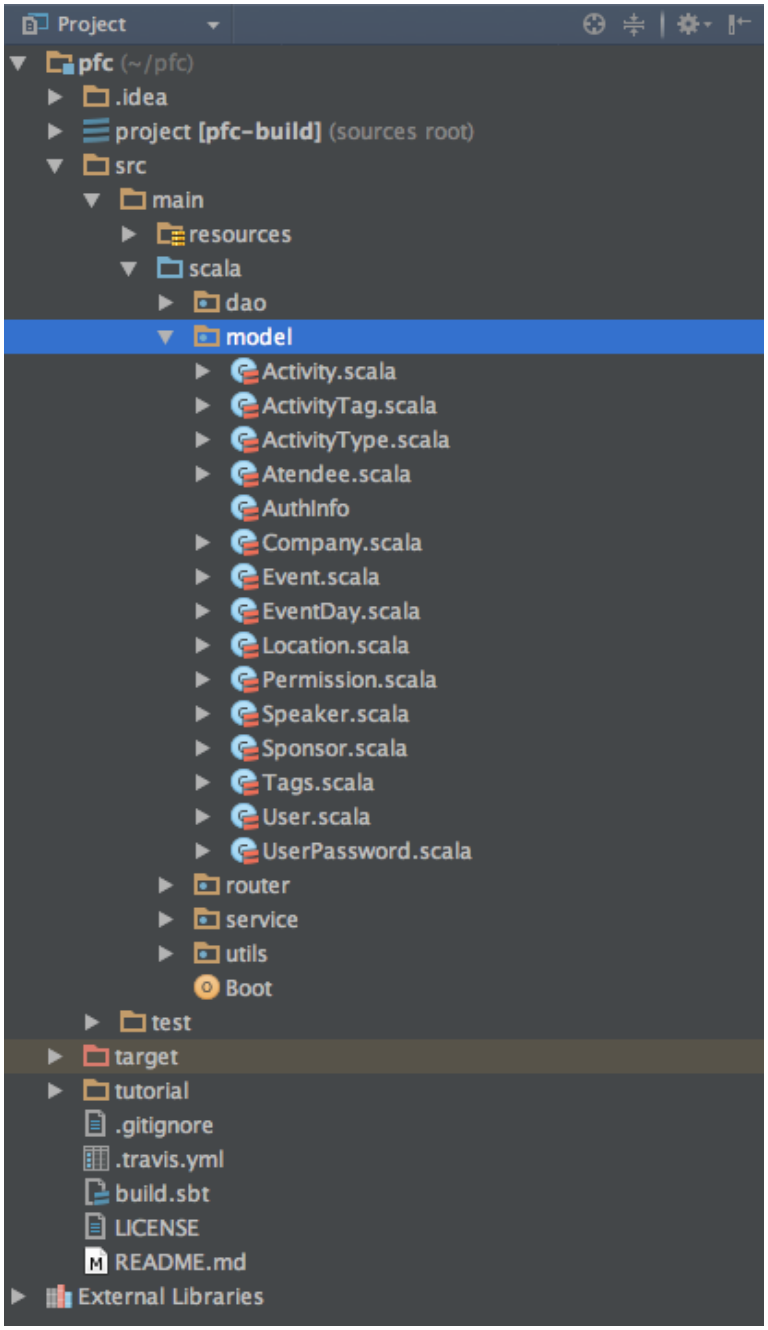


Figura 3 - Paquete Model

2.3. Paquete Service

El paquete *service* (Figura 4 - Paquete Service) es el encargado de orquestar los dao y proveer las transformaciones necesarias entre los objetos que vienen desde la vista para ser insertados en la base de datos.

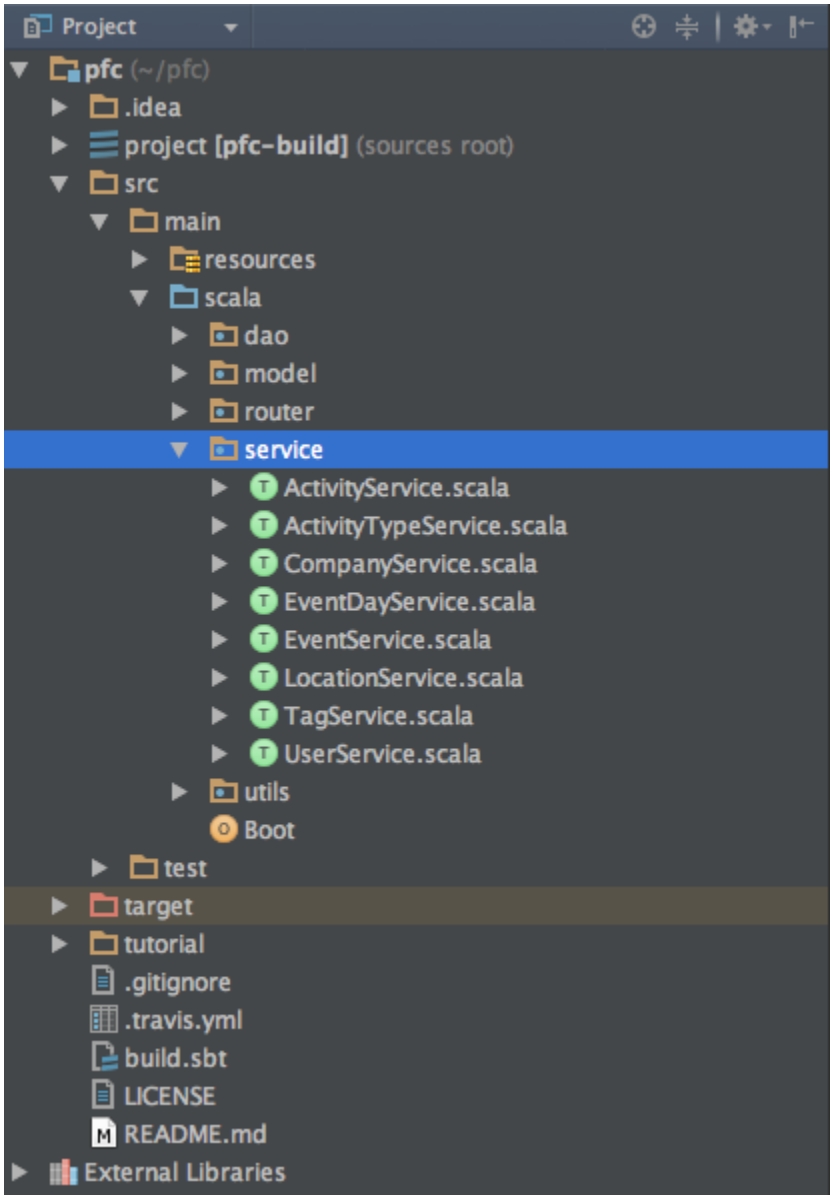


Figura 4 - Paquete Service

Paquete router

El paquete *router* (Figura 5 - Paquete Router) es el encargado de definir los distintos REST endpoints y proveer las distintas respuestas a el cliente (Códigos HTTP, mensajes de error, etc.).

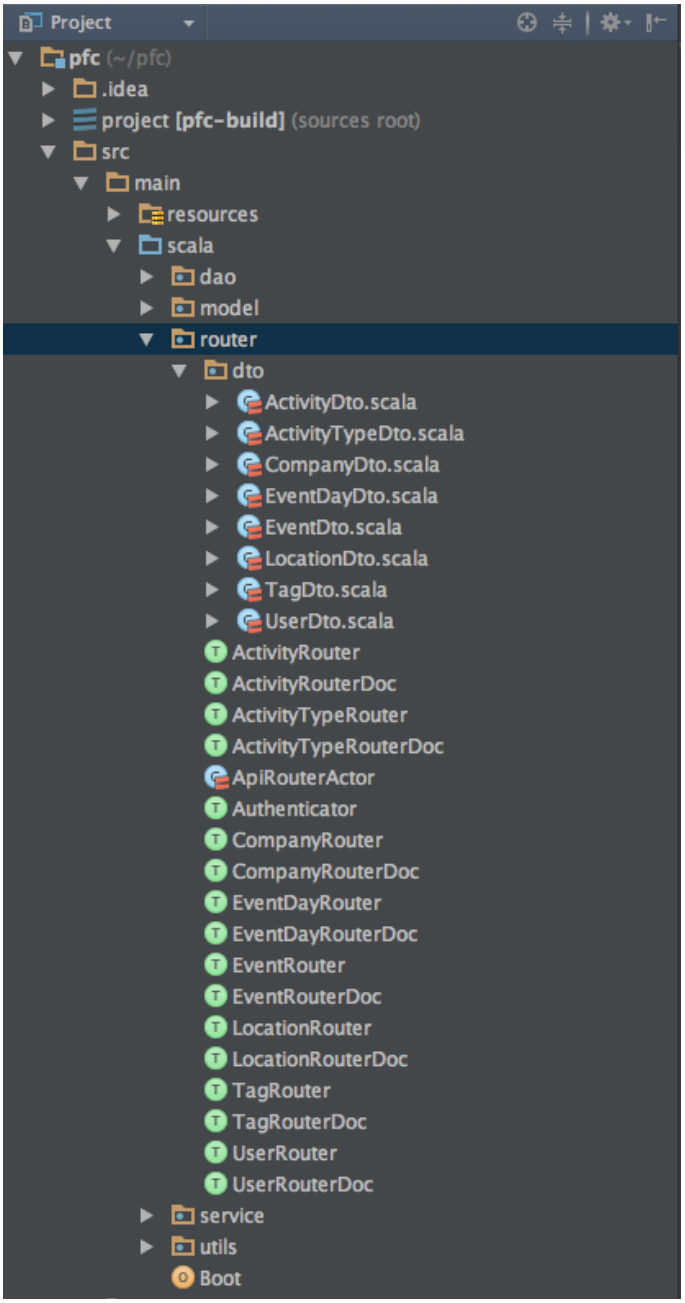


Figura 5 - Paquete Router

2.4. Paquete *utils*

El paquete *utils* (Figura 6 - Paquete Utils) contiene una serie de clases que sirven como objetos auxiliares para diversos usos, como puede ser un conversor de objetos JSON o el objeto que posee todas las referencias a los *strings* de configuración.

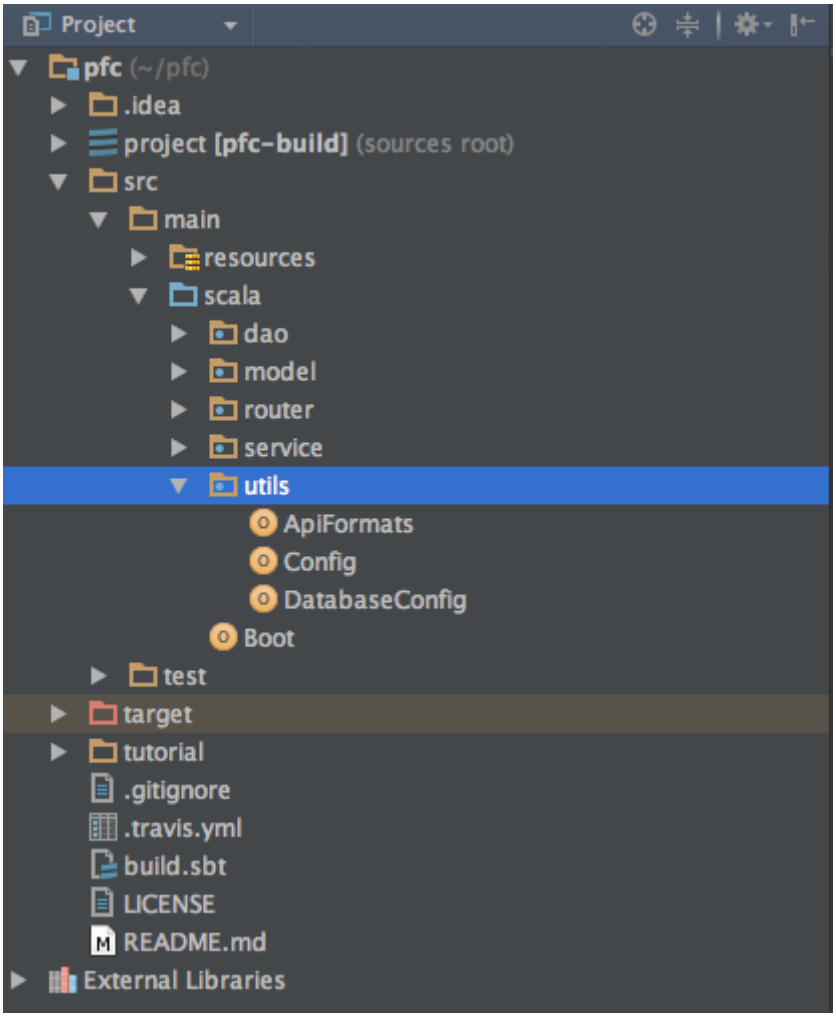


Figura 6 - Paquete Utils

3. El framework Spray

Vamos a pasar a explicar el código de la aplicación de una manera más detallada, en primer lugar se puede observar el fichero encargado de definir las dependencias y la meta información de la aplicación, este fichero denominado build.sbt contiene la siguiente información.

```
import NativePackagerKeys._

packageArchetype.java_application

version       := "0.1"

scalaVersion  := "2.11.6"

scalacOptions := Seq("-unchecked", "-deprecation", "-encoding", "utf8")

libraryDependencies ++= {
  val akkaVersion = "2.3.6"
  val sprayVersion = "1.3.2"
  Seq(
    "io.spray"            %%   "spray-can"       % sprayVersion,
    "io.spray"            %%   "spray-routing"   % sprayVersion,
    "io.spray"            %%   "spray-json"      % sprayVersion,
    "io.spray"            %%   "spray-testkit"   % sprayVersion  % "test",
    "com.typesafe.akka"   %%   "akka-actor"      % akkaVersion,
    "com.typesafe.akka"   %%   "akka-testkit"    % akkaVersion   % "test",
    "org.specs2"          %%   "specs2-core"     % "2.3.11"      % "test",
    "com.typesafe.slick"  %%   "slick"           % "3.0.0",
    "org.slf4j"            %  "slf4j-nop"       % "1.7.7",
    "org.slf4j"            %  "slf4j-api"       % "1.7.7",
    "org.scalatest"        %   "scalatest_2.11" % "2.2.1" % "test",
    "com.typesafe"         %   "config"         % "1.2.1",
    "postgresql"           %   "postgresql"     % "9.1-901.jdbc4",
    "com.h2database"       %   "h2"             % "1.4.182",
    "com.gettyimages"      %%  "spray-swagger"  % "0.5.0",
    "org.webjars"          %   "swagger-ui"     % "2.0.12",
    "com.github.t3hnar"    %%  "scala-bcrypt"   % "2.4",
    "org.mindrot"          %   "jbcrypt"        % "0.3m",
    "joda-time"            %  "joda-time"       % "2.7",
    "org.joda"             %  "joda-convert"    % "1.7",
    "com.github.tototoshi" %% "slick-joda-mapper" % "2.0.0"
  )
}

fork in Test := false

parallelExecution in Test := false

Revolver.settings
```

Figura 7 - build.sbt

Los puntos más importantes son:

-Versionado y opciones de compilación

```
version       := "0.1"

scalaVersion  := "2.11.6"

scalacOptions := Seq("-unchecked", "-deprecation", "-encoding", "utf8")
```

Figura 8 - Variables de versionado

En la Figura 8 - Variables de versionado, se observa como se define la versión de la aplicación "*version"*, la versión de Scala usada "*scalaVersion"*, en este caso 2.11.6, siendo la última disponible en el momento y por último "*scalacOptions"* estas son opciones que queremos que SBT incluya durante la compilación.

La versión de Scala es importante a la hora de hacer uso de librerías externas ya que muchas de ellas solo están disponibles para determinadas versiones de Scala.

–Dependencias

En la Figura 9 – Dependencias, se muestra la definición de las dependencias del proyecto. Tenemos dos opciones a la hora de depender de librerías de terceros una de ellas es descargar el Jar necesario y ponerlo en la carpeta /lib (no estarían supervisadas) lo que las añade al classpath, o podemos dejar que SBT las gestione. Al igual que otros gestores de dependencias como Maven, o Gradle, SBT proporciona una manera sencilla de obtener librerías que son obtenidas normalmente de un repositorio central.

Las librerías que se observan en la Figura 10 - Dependencias del framework, serian el núcleo de la aplicación comparten la versión por lo que se puede extraer como si fuera una variable dentro del código, la etiqueta "test" de la librería "spray-testkit" especifica que solo queremos usar esta librería para el código de test (src/test/scala) y no en el código de la aplicación en sí.

```
libraryDependencies ++= {
  val akkaVersion = "2.3.6"
  val sprayVersion = "1.3.2"
  Seq(
    "io.spray"            %%   "spray-can"       % sprayVersion,
    "io.spray"            %%   "spray-routing"   % sprayVersion,
    "io.spray"            %%   "spray-json"      % sprayVersion,
    "io.spray"            %%   "spray-testkit"   % sprayVersion  % "test",
    "com.typesafe.akka"   %%   "akka-actor"      % akkaVersion,
    "com.typesafe.akka"   %%   "akka-testkit"    % akkaVersion   % "test",
    "org.specs2"          %%   "specs2-core"     % "2.3.11"      % "test",
    "com.typesafe.slick"  %%   "slick"           % "3.0.0",
    "org.slf4j"            %  "slf4j-nop"       % "1.7.7",
    "org.slf4j"            %  "slf4j-api"       % "1.7.7",
    "org.scalatest"        %   "scalatest_2.11" % "2.2.1" % "test",
    "com.typesafe"         %   "config"         % "1.2.1",
    "postgresql"           %   "postgresql"     % "9.1-901.jdbc4",
    "com.h2database"       %   "h2"             % "1.4.182",
    "com.gettyimages"     %%   "spray-swagger"  % "0.5.0",
    "org.webjars"          %   "swagger-ui"     % "2.0.12",
    "com.github.t3hnar"   %%   "scala-bcrypt"   % "2.4",
    "org.mindrot"          %   "jbcrypt"        % "0.3m",
    "joda-time"            %  "joda-time"       % "2.7",
    "org.joda"             %  "joda-convert"    % "1.7",
    "com.github.tototoshi" %% "slick-joda-mapper" % "2.0.0"
  )
}
```

Figura 9 – Dependencias

```
    "io.spray"            %%   "spray-can"       % sprayVersion,
    "io.spray"            %%   "spray-routing"   % sprayVersion,
    "io.spray"            %%   "spray-json"      % sprayVersion,
    "io.spray"            %%   "spray-testkit"   % sprayVersion  % "test",
```

Figura 10 - Dependencias del framework

```
20      "com.typesafe.akka"   %%   "akka-testkit"    % akkaVersion   % Test,
21      "org.specs2"          %%   "specs2-core"     % "2.3.11"      % Test,
22      "org.scalatest"        %   "scalatest_2.11"  % "2.2.1"       % Test,
```

Figura 11 - Dependencias de tests

En la Figura 11 - Dependencias de tests, se muestran las librerías encargadas de proveer una API para el desarrollo de test, *specs2* nos proporciona una DSL intuitiva y que ayuda a denominar y anidar tests dependiendo del comportamiento que deseamos testear.

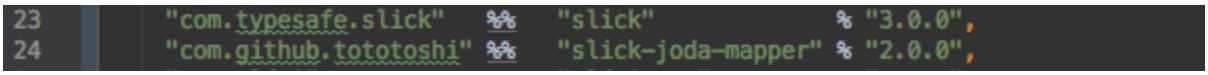


Figura 12 - Dependencias del ORM

Slick nos proporciona una funcionalidad asociada al acceso a la base de datos, se podría denominar ORM, en la versión 3 nos da la posibilidad de crear un código totalmente asíncrono para el I/O con la BBDD lo que esta considerado el modelo a seguir dentro de la programación hoy en día (Figura 12 - Dependencias del ORM). Además, *Slick* provee diferentes drivers para múltiples base de datos lo que nos da la posibilidad de tener un único desarrollo y la posibilidad de cambiar la base de datos de una manera muy sencilla.

Slick-joda-mapper provee unos transformadores para fechas en la base de datos y las clases usadas que las representan en el código.

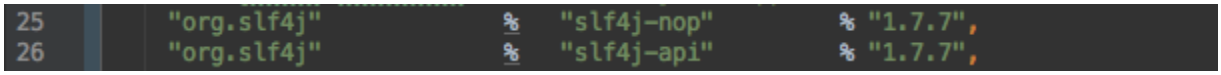


Figura 13 - Dependencias de logs

Las librerías encargadas del sistema de logs (Figura 13 - Dependencias de logs), nos dan la posibilidad de loguear con distintos niveles, estos niveles (DEBUG, INFO, ERROR,...) pueden ser administrados en un archivo de configuración en el que se selecciona que nivel deseamos ver en los logs, lo que nos proporcionaría ese nivel y los inferiores.

Figura 14 - Dependencias de la base de datos

Estas librerías proporcionan una implementación de *JDBC* para *postresql* y para *h2*, ambas bases de datos SQL, h2 crea una base datos en memoria, lo que permite realizar test de integración con una alta similitud al entorno de producción.

```
30      "com.gettyimages"      %%   "spray-swagger"     % "0.5.0",
31      "org.webjars"           %   "swagger-ui"        % "2.0.12",
```

Figura 15 - Dependencias de swagger

Swagger nos proporciona una manera elegante y sencilla para escribir la documentación de la API rest, desde el punto de vista del desarrollador, tener acceso a una documentación completa de la api es un valor diferenciador a la hora de elegir una aplicación contra la que desarrollar.

4. Punto de entrada de la aplicación (~MainClass)

Ahora vamos a crear el fichero encargado de arrancar la aplicación, la misión de este fichero es levantar el servidor de spray (*Spray-Can*), este fichero, llamado Boot.scala (Figura 16 - Boot.scala) extiende el trait App de la librería Scala (conocido como main class en Java).

```scala
import akka.actor.{ ActorRef, ActorSystem, Props }
import akka.io.IO
import akka.pattern.ask
import akka.util.Timeout
import router.ApiRouterActor
import service._
import spray.can.Http
import utils.Config._

import scala.concurrent.duration._


object Boot extends App {

  // we need an ActorSystem to host our application in
  implicit val system = ActorSystem(app.systemName)

  // create and start our service actor
  val userActor: ActorRef = system.actorOf(
    Props(
      classOf[ApiRouterActor],
      UserService,
      ActivityService,
      EventService,
      LocationService,
      CompanyService,
      TagService,
      EventDayService,
      ActivityTypeService
    ), app.userServiceName)

  implicit val timeout = Timeout(5.seconds)
  // start a new HTTP server on port 8080 with our service actor as the handler
  IO(Http) ? Http.Bind(userActor, interface = app.interface, port = app.port)

}
```

Figura 16 - Boot.scala

5. Actor Recepcionista

Ahora pasamos a echar un vistazo al actor encargado de recibir las peticiones HTTP que vienen del servidor, ApiRouterActor.Scala. Vamos a dividir la clase en tres partes para así explicar mas detalladamente sus funciones:

```scala
package router

import ...


// we don't implement our route structure directly in the service actor because
// we want to be able to test it independently, without having to spin up an actor
class ApiRouterActor(
  userServ: UserService,
  activityServ: ActivityService,
  eventServ: EventService,
  locationServ: LocationService,
  companyServ: CompanyService,
  tagServ: TagService,
  eventDayServ: EventDayService,
  activityTypeServ: ActivityTypeService
) extends Actor
  with UserRouter
  with ActivityRouter
  with EventRouter
  with LocationRouter
  with CompanyRouter
  with TagRouter
  with EventDayRouter
  with ActivityTypeRouter
  with ActorLogging
  with Authenticator
{

  override val userService = userServ
  override val activityService = activityServ
  override val eventService = eventServ
  override val locationService = locationServ
  override val companyService = companyServ
  override val tagService = tagServ
  override val eventDayService = eventDayServ
  override val activityTypeService = activityTypeServ

```

Figura 17 - ApiRouterActor.scala

Lo más importante a remarcar aquí es que nuestra clase extiende el trait (similar a las interfaces o clases abstractas de Java) Actor.scala del framework *Akka*, además de

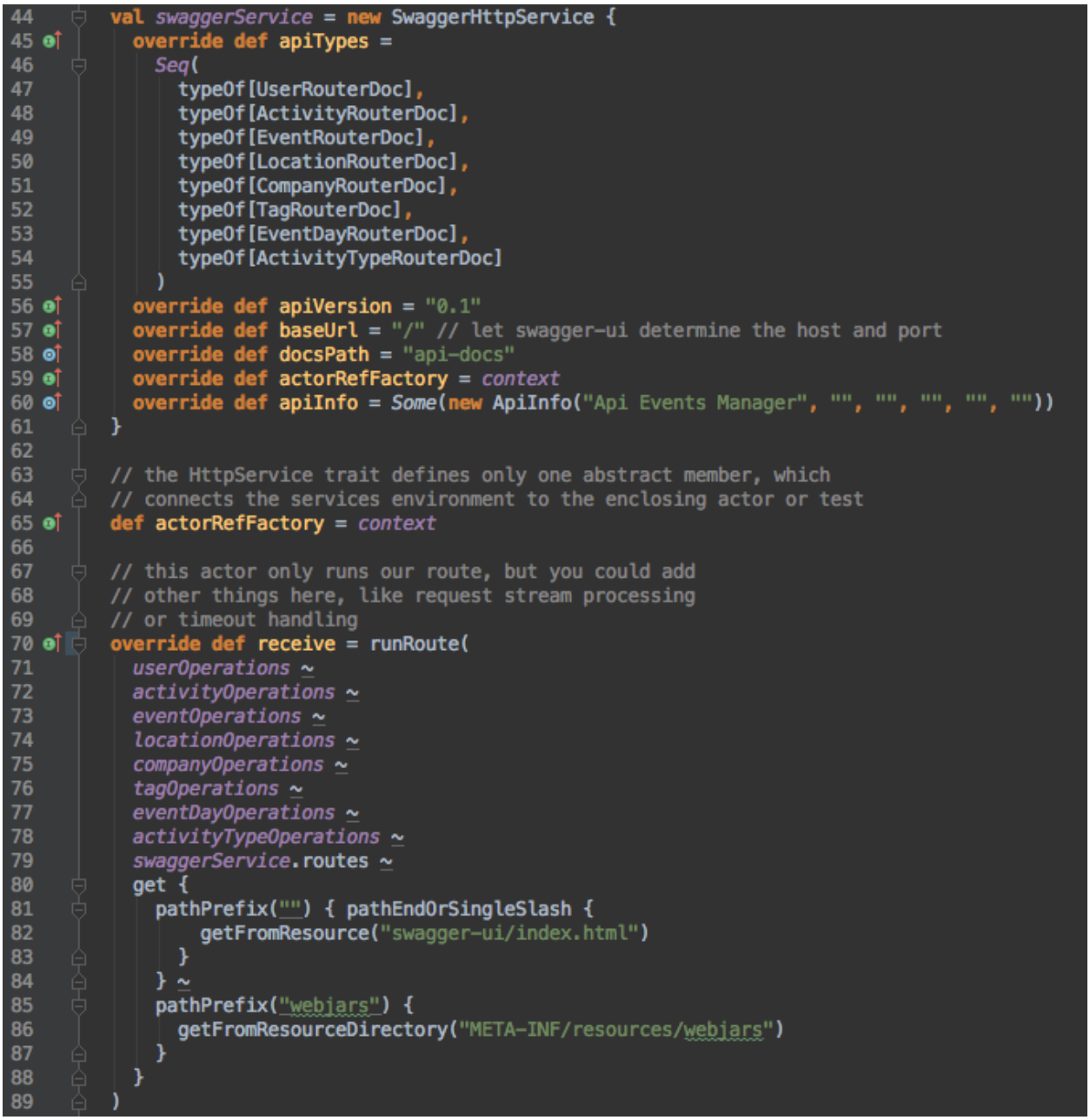
señalar los distintos servicios que necesita el actor para ser instanciado en una determinada factoría.

Como se puede ver también mezclamos en la clase los distintos traits que contienen la la manera de resolver distintas peticiones http, por ejemplo el trait encargado de responder HTTP GET /users/1 sería UserRouter.scala y el encargado de responder HTTP GET /activities seria ActivityRouter.scala y así respectivamente con los demás endpoints que podemos encontrar en la documentación de la API.

También sobrescribimos los servicios que han sido inyectados en el actor, estos servicios son usados en cada trait de cada entidad para procesar la petición http, volviendo al ejemplo HTTP GET /activities el método encargado de manejar la petición HTTP haría una llamada al método del ActivityService.scala que obtiene todas las actividades.

En la Figura 18 - Creación de swagger y anidación de los endpoints, se observa la instanciación del servicio Swagger, se añaden los distintos traits que incluyen la meta información asociada a los endpoints (UserRouterDoc.Scala,…), con esta información swagger genera la documentación, además como se observa se añade un *endpoint* en el método *receive* que para determinada request en este caso yendo únicamente al dominio *pathPrefix("")*

(lanzando la aplicación en local seria http://localhost:8080), como se puede ver también se crea un *endpoint* para subministrar los recursos necesario para levantar la aplicación de swagger, esta aplicación se ejecuta en el lado del cliente como una SPA (Single Page Application) que únicamente se comunica con nuestro servidor mediante JSON, por lo que se delega la generación del HTML al navegador.



```
val swaggerService = new SwaggerHttpService {
  override def apiTypes =
    Seq(
      typeOf[UserRouterDoc],
      typeOf[ActivityRouterDoc],
      typeOf[EventRouterDoc],
      typeOf[LocationRouterDoc],
      typeOf[CompanyRouterDoc],
      typeOf[TagRouterDoc],
      typeOf[EventDayRouterDoc],
      typeOf[ActivityTypeRouterDoc]
    )
  override def apiVersion = "0.1"
  override def baseUrl = "/" // let swagger-ui determine the host and port
  override def docsPath = "api-docs"
  override def actorRefFactory = context
  override def apiInfo = Some(new ApiInfo("Api Events Manager", "", "", "", "", ""))
}

// the HttpService trait defines only one abstract member, which
// connects the services environment to the enclosing actor or test
def actorRefFactory = context

// this actor only runs our route, but you could add
// other things here, like request stream processing
// or timeout handling
override def receive = runRoute(
  userOperations ~
  activityOperations ~
  eventOperations ~
  locationOperations ~
  companyOperations ~
  tagOperations ~
  eventDayOperations ~
  activityTypeOperations ~
  swaggerService.routes ~
  get {
    pathPrefix("") { pathEndOrSingleSlash {
        getFromResource("swagger-ui/index.html")
    }
    } ~
    pathPrefix("webjars") {
      getFromResourceDirectory("META-INF/resources/webjars")
    }
  }
)
```

Figura 18 - Creación de swagger y anidación de los endpoints

6. Definiendo un endpoint

Vamos a explicar paso a paso como fue definido el endpoint para los usuarios (/users). Para empezar necesitamos la clase que represente la entidad Usuario.

```
9   @ApiModel(description = "An User entity")
10  case class User(
11    @(ApiModelProperty @field)(value = "unique identifier for the user")
12    id: Int,
13
14    @(ApiModelProperty @field)(value = "email of the user")
15    email: String,
16
17    @(ApiModelProperty @field)(value = "user's first name")
18    firstName: Option[String] = None,
19
20    @(ApiModelProperty @field)(value = "user's last name")
21    lastName: Option[String] = None,
22
23    @(ApiModelProperty @field)(value = "user's twitter id")
24    twitterId: Option[String] = None,
25
26    @(ApiModelProperty @field)(value = "user's linkedin id")
27    linkedinId: Option[String] = None,
28
29    @(ApiModelProperty @field)(value = "user's bio")
30    bio: Option[String] = None,
31
32    @(ApiModelProperty @field)(value = "user permission")
33    permission: String,
34
35    @(ApiModelProperty @field)(hidden = true)
36    passwordId: Option[Int] = None
37  )
38  object User extends DefaultJsonProtocol{
39    implicit val userFormat = jsonFormat9(User.apply)
40  }
41
```

Figura 19 - User.scala

Como se observa la clase que define la entidad es un case class, este contiene los atributos que se le quiere dar al objeto además se le puede dar valores por defecto si no se proveen valores para un determinado atributo durante la instanciación, también se anota mediante las anotaciones @ApiModel and @ApiModelProperty esto provee meta información que va a servir como documentación para el servicio Swagger.

Podemos ver un punto importante como es el denominado "*companion object*" de la entidad "*object User extends DefaultJsonProtocol*" con esta simplicidad estamos generando algo que lleva dando quebraderos de cabeza a desarrolladores desde los albores de la computación, la serialización, en este caso a Json. Este "*implicit val*" provee al compilador la "receta" de cómo debe serializar/deserializar la entidad User a Json o viceversa.

Ahora definimos el trait denominado UserRouter:

```scala
package router

import ...


// this trait defines our service behavior independently from the service actor
trait UserRouter extends HttpService with UserRouterDoc {
  self: Authenticator =>

  val userService: UserService

  val userOperations: Route = postRouteUser ~ readRouteUser ~ readAllRouteUser ~ deleteRouteUser

  override def readRouteUser = path("users" / IntNumber) { userId =>
      get {
        authenticate(basicUserAuthenticator) { authInfo =>
          respondWithMediaType(`application/json`) {
            onComplete(userService.get(userId)) {
              case Success(Some(user)) => complete(user)
              case Success(None) => complete(NotFound, "User not found")
              case Failure(ex) => complete(InternalServerError, s"An error occurred: ${ex.getMessage}")
            }
          }
        }
      }
    }
```

Figura 20 - UserRouter.scala

Este trait extiende *HttpService*, *UserRouterDoc* y con tiene un denominado "self-type" con el trait Authenticator, esto significa que si queremos mezclar el trait UserRouter debemos también incluir el trait Authenticator, es decir hemos generado una dependencia entre UserRouter y Authenticator. Se ha definido el router independiente del actor ya que así podemos mezclar este trait con un actor de test lo que nos evitará tener que crear nosotros el sistema de actores únicamente para los tests.

Se define el service, **val userService: UserService** de manera que cuando se extienda el trait se deba proveer de un objeto del tipo UserService en userOperations

```
val userOperations: Route = postRouteUser ~ readRouteUser ~ readAllRouteUser ~ deleteRouteUser
```

Agregamos los diferentes métodos encargados (el símbolo ~ es un método “infix”) de gestionar las peticiones HTTP, estos métodos son los que luego se añaden en el método *receive* del ApiRouterActor.Scala.

Vamos a pasar a explicar el método “*override def readRouteUser*” parte por parte.

En primer lugar se especifica cual va a ser la URI en el servidor

```
path("users" / IntNumber) { userId =>
```

arrancando la aplicación en local la uri seria http://localhost:8080/users/3 (3 como userId que queremos obtener), “userId =>” hace que tengamos el id del usuario de la request dentro del scope del método con el nombre “userId”, este será pasado al servicio como argumento del método .get(id: Int).

```
get {
 authenticate(basicUserAuthenticator) { authInfo =>
```

El método “get” indica que la petición que vamos a tratar es una de los métodos HTTP (GET, POST, DELETE, etc).

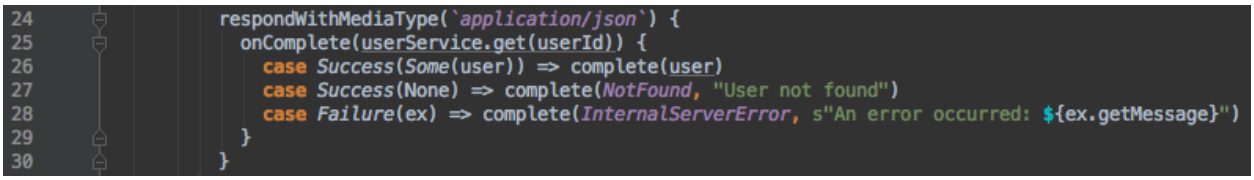


Figura 21 - Respuesta del router /user/{userId} (UserRouter.scala)

respondWithMediaType(`application/json`) {

Indica que en la respuesta de la petición se va a indicar el MediaType como 'application/json'

onComplete(userService.get(userId)) {

Hace la llamada al método get del servicio, este método es asíncrono, devuelve un Future[Option[User]] por lo que tenemos que "desenvolver" el objeto Future por lo que usamos el método onComplete, que realiza esta función y "envuelve" el resultado en el tipo Try[T] que lo resolvemos haciendo *pattern matching* con las siguientes opciones:

case Success(Some(user)) => complete(user)

Hemos obtenido la instancia del tipo Some[User] por lo que devolvemos en la respuesta el user.

case Success(None) => complete(NotFound, "User not found")

En este caso a pesar de no existir ningún error, obtenemos el tipo None, por lo que se devuelve en la petición con un 404 indicando que no existe un usuario con ese ID en el sistema, por ultimo

case Failure(ex) => complete(InternalServerError, s"An error occurred: ${ex.getMessage}")

Nos indica que el resultado del servicio ha finalizado con una excepción por lo que devolvemos un 500 con una explicación del error. Los demás métodos, *readAllRouteUser* y *deleteRouteUser* siguen una estructura similar excepto *postRouteUser* que tiene un paso intermedio encargado de procesar el JSON.

```
63  override def postRouteUser: Route = path("users") {
64    post {
65      authenticate(basicUserAuthenticator) { authInfo =>
66        authorize(authInfo.hasPermissions("ADMIN")) {
67          entity(as[UserDto]) { user =>
68            respondWithMediaType(`application/json`) {
69              onComplete(userService.add(user)) {
70                case Success(Some(newUser)) => complete(Created, newUser)
71                case Success(None) => complete(NotAcceptable, "Invalid user")
72                case Failure(ex) => complete(InternalServerError, s"An error occurred: ${ex.getMessage}")
73              }
74            }
75          }
76        }
77      }
78    }
79  }
```

Figura 22 - Operacion Http POST (UserRouter.scala)

Además de un nivel más restrictivo de permiso, ahora es “ADMIN”, se observa como se procesa el json con ***entity(as[UserDto]) { user =>*** y se inyecta el UserDto (clase que representa un usuario que se obtiene desde la request) que luego se transforma en User normal para su inserción en la base de datos dentro del método *.add* del servicio.

7. Documentación del Router

La documentación que es generada por Swagger esta definida en un trait que es extendido por el propio router de cada entidad, en este caso correspondería el UserRouterDoc.scala

```scala
// Trying to not pollute the code with annotations
@Api(value = "/users", description = "Operations for users.", consumes= "application/json",  produces = "application/json")
trait UserRouterDoc {
```

Figura 23 - Declaración de la clase UserRouterDoc.scala

Como se observa en el comentario se ha decidido por implementar las anotaciones con los comentarios en un trait por separado para poder de esta manera tener por una parte la implementación del router y por otra la meta información.

Con la anotación @Api especificamos el nombre del *endpoint* además del tipo de datos que consume y que produce, en este caso Json.

```scala
    @ApiOperation(value = "Get a user by id", httpMethod = "GET", response = classOf[User])
    @ApiImplicitParams(Array(
      new ApiImplicitParam(name = "userId", value="ID of the user that needs to retrieved", required = true, dataType = "integer", paramType = "path" )
    ))
    @ApiResponses(Array(
      new ApiResponse(code = 200, message = "Ok"),
      new ApiResponse(code = 404, message = "User not found")
    ))
    def readRouteUser: Route
```

Figura 24 - Documentación del método readRouteUser (UserRouterDoc.scala)

@ApiOperation

Describe la acción de la petición HTTP, el método, y la respuesta.

@ApiImplicitParams

Describe los atributos que se pasan en la petición. En este caso únicamente el userId, también se especifican el tipo del argumento, y como va este dentro de la request.

@ApiResponses

Las posibles opciones que se pueden obtener como respuesta desde el servidor.

8. Tests de la aplicación

Todo buen desarrollo debe ir acompañado de test, unitarios, de integración, de regresión, etc. En la practica se realizan tests de integración haciendo “mock” de la base de datos.

```
20  class UserIntegrationSpec extends Specification with Specs2RouteTest with UserRouter with SpecSupport with Authenticator with Mockito {
21
22    // connects the DSL to the test ActorSystem
23    implicit def actorRefFactory = system
24
25    override val userService = UserService
26
27    val userAdmin = BasicHttpCredentials("test1@test.com", "password1")
28    val userNotAdmin = BasicHttpCredentials("test2@test.com", "password1")
```

Figura 25 - Definición de UserIntegrationSpec.scala

Se observa como en la creación de la clase de test, se mezclan una serie traits los más importantes son Specs2RouteTest.scala y el trait que queremos testear UserRouter.scala. SpecSupport es un trait propio encargado de crear una base de datos en memoria (H2) insertando además unos datos de tests que van a ser los que luego se comprueban en el router. Authenticator es el trait encargado de la autentificación.

Declaramos también un usuario que en la base de datos tiene permisos de admin y otro que no con val userAdmin y val userNotAdmin respectivamente, vamos a usar estos usuarios para los test ya que son necesarios para acceder a los endpoint, el userAdmin

tiene acceso a las operaciones de escritura (POST, DELETE), mientras que el userNotAdmin no.

```
46    "return a single user for GET requests to users path" in this {
47      Get("/users/1") ~> addCredentials(userAdmin) ~> userOperations ~> check {
48        responseAs[User] === DatabaseSupportSpec.users.head
49      }
50    }
51
52    "return the id for DELETE requests to users path" in this {
53      Delete("/users/1") ~> addCredentials(userAdmin) ~> userOperations ~> check {
54        status mustEqual StatusCodes.OK
55        val result: Seq[User] = Await.result(db.run{UserDao.getAll}, Duration.Inf)
56        result mustEqual DatabaseSupportSpec.users.filterNot(_.id == 1)
57      }
58    }
```

Figura 26 - Definición de tests (UserIntegrationSpec.scala)

La manera de escribir los test es muy intuitiva la librería Specs2 nos proporciona una DSL orientada al BDD (Behaviour Driven Development).

Ingeniería Técnica Informática de Gestión

PROYECTO FIN DE CARRERA

ANEXO II : MANUAL DEL USUARIO (ESTUDIO DEL STACK TECNOLÓGICO PARA EL DESARROLLO DE UN FRAMEWORK ESCALABLE ORIENTADO A APIS REST)

Autor: Giancarlo Alfredo Muñoz Reinoso
Tutor: Alejandro Baldominos Gómez

Leganés, Septiembre 2015

Tabla de contenidos

Índice de figuras

1. Introducción

Este manual está orientado al usuario de la API[1] , este siempre va a ser un desarrollador, por lo que se presupone cierto conocimiento de la tecnología REST[2] así como nociones de HTTP.

Se requiere que el administrador le suministre un usuario y una contraseña para poder acceder a los recursos, así como para disponer de permisos de escritura (crear, borrar y modificar entidades).

Se va a emplear como cliente REST y para facilitar las explicaciones la aplicación web Swagger que nos proprociona una completa documentación y una sencilla interfaz con la que enviar peticiones al servidor y poder visualizar las respuestas.

2. Interfaz web Swagger

Vamos a pasar a explicar la pantalla inicial (Figura 1 - Pantalla inicial Swagger), como se puede observar es posible visualizar todos los recursos de la aplicación (*users*, *activities*, etc.) , no es necesario tener usuario en la aplicación para ver el formato con el que hay que realizar las peticiones ni tampoco las respuestas que el servidor manda.

Podemos expandir las operaciones para cada entidad para poder las distintas operaciones que admiten las URI[i].

Vamos a explicar las partes principales que se muestran en la aplicación para una operación sobre un recurso.

---

[1] API: *Aplication Programming Interface*, “contrato” que una librería/servicio web establece con los usuarios de esta, en la api se expone las operaciones disponibles.

[2] REST: *Representational State Transfer*, diseño arquitectónico de un sistema en el que cada *Uniform Resource Identifier* representa una entidad del modelo.

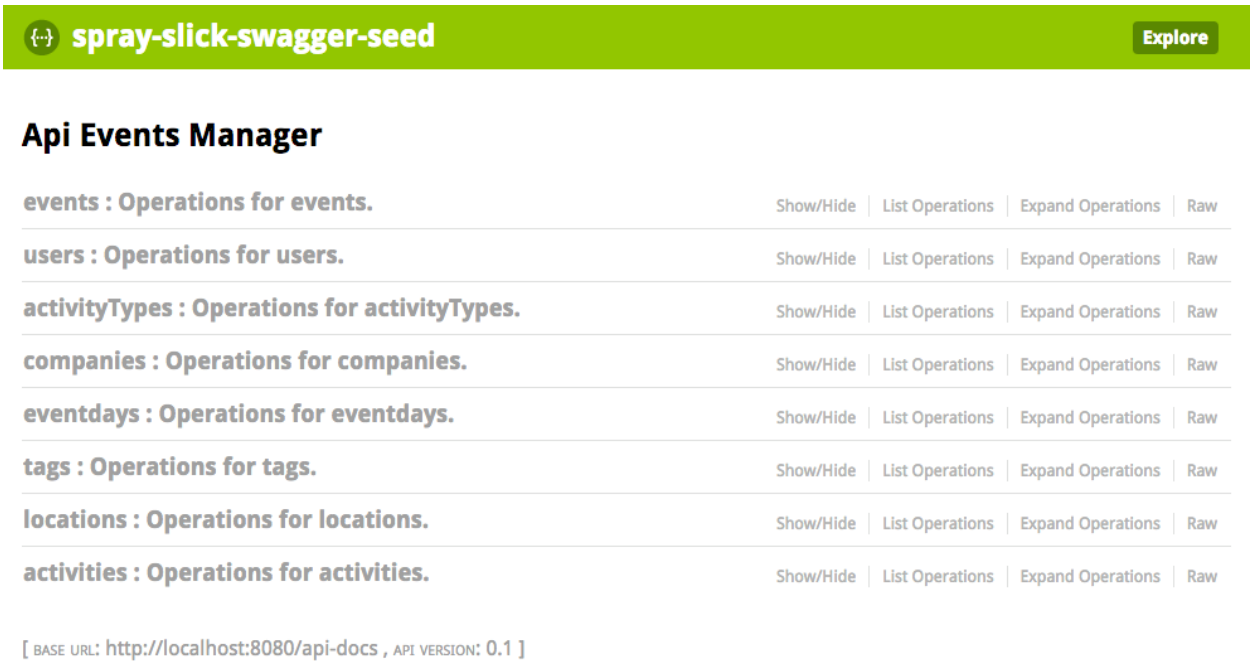


Figura 1 - Pantalla inicial Swagger

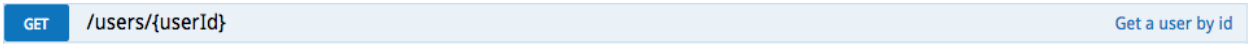


Figura 2 - Cabecera de la petición

En la Figura 2 - Cabecera de la petición, observamos la URI, el método HTTP usado y una pequeña explicación de la acción que realiza esa operación.

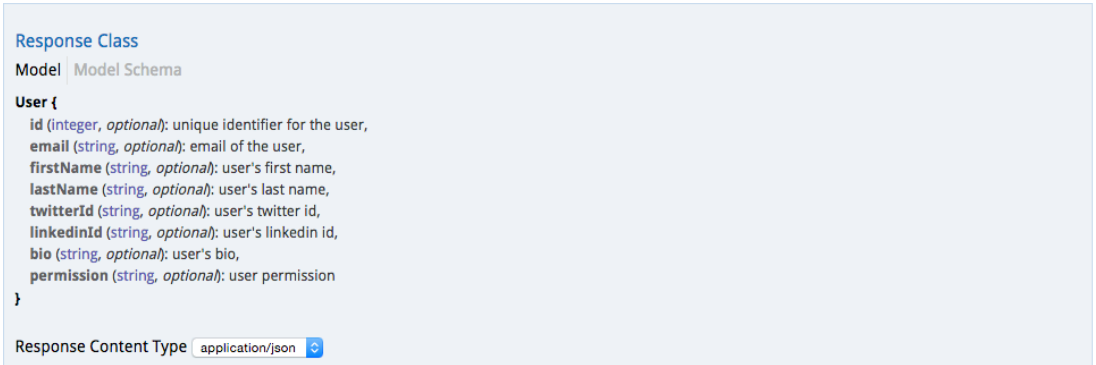
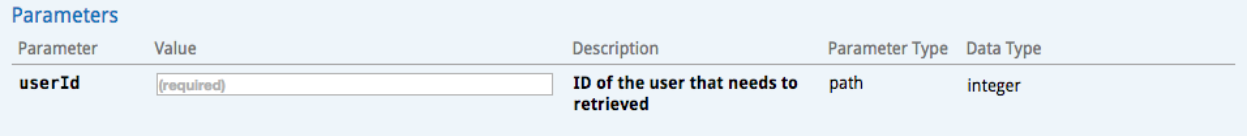


Figura 3 - Respuesta de la petición

En la Figura 3 - Respuesta de la petición, se puede ver el modelo de la respuesta así como sus campos y la obligatoriedad o no de los mismos, también se puede especificar el formato de la respuesta, en nuestro caso únicamente disponemos de JSON.

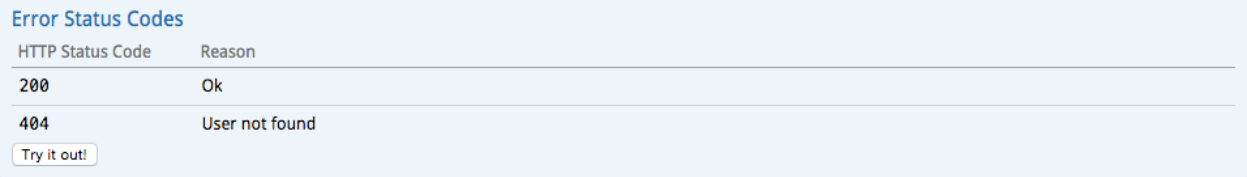


Parameters

Parameter	Value	Description	Parameter Type	Data Type
userId	(required)	**ID of the user that needs to retrieved**	path	integer

Figura 4 - Parámetros de la petición

En la Figura 4 - Parámetros de la petición se explican los parámetros que requiere la petición además de la descripción del parámetro, cómo va a ir este parámetro en la petición y el tipo de dato del mismo.



Error Status Codes

HTTP Status Code	Reason
200	Ok
404	User not found

Try it out!

Figura 5 - Códigos de respuesta

En la Figura 5 - Códigos de respuesta, se muestran los códigos de respuesta HTTP obtenidos del servidor.

3. Autorización

No es necesario usar Swagger para comunicarse con el servidor, pero hay que tener en cuenta sin embargo que la petición HTTP debe incluir el header “Authorization”, la cabecera tiene el siguiente formato:

1. Email y contraseña se combinan en un una única cadena *“usuario:contraseña”* , el usuario no puede contener el carácter “:”

2. La cadena resultante se codifica en Base64 (Gonzalez, A.)

3. El método de autorización y la cadena con un espacio entre estos dos.

Ejemplo:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Como se puede observar toda la documentación está escrita en la lengua de Shakespeare, por lo que se van a emplear los apelativos ingleses de las entidades en este manual.

Dentro de cada operación se observa un botón denominado “*Try it out*”, este botón ejecuta la acción, es decir realiza la petición HTTP desde el navegador al servidor de la aplicación.

Si es la primera vez que lanzamos una petición saltará un pop-up preguntando por el usuario y la contraseña, como se ha explicado anteriormente la operación de

serialización de nuestro usuario y contraseña, es realizada por *Swagger* internamente en el navegador por lo que no hay que preocuparse por ello.

4. Petición de ejemplo

Vamos a realizar el proceso completo de obtener datos del servidor a través de Swagger, escogemos como ejemplo *"/events/{eventId}/sponsors"* este endpoint nos demuestra la riqueza de una arquitectura *REST,* en este caso queremos obtener los sponsors del evento con identificador "1", como se puede observar un *Sponsor* es simplemente una relación entre una compañía y un evento, por lo que en la respuesta obtenemos una entidad de lista de la entidad "Company" (Figura 6 - "GET" sponsors para un evento).

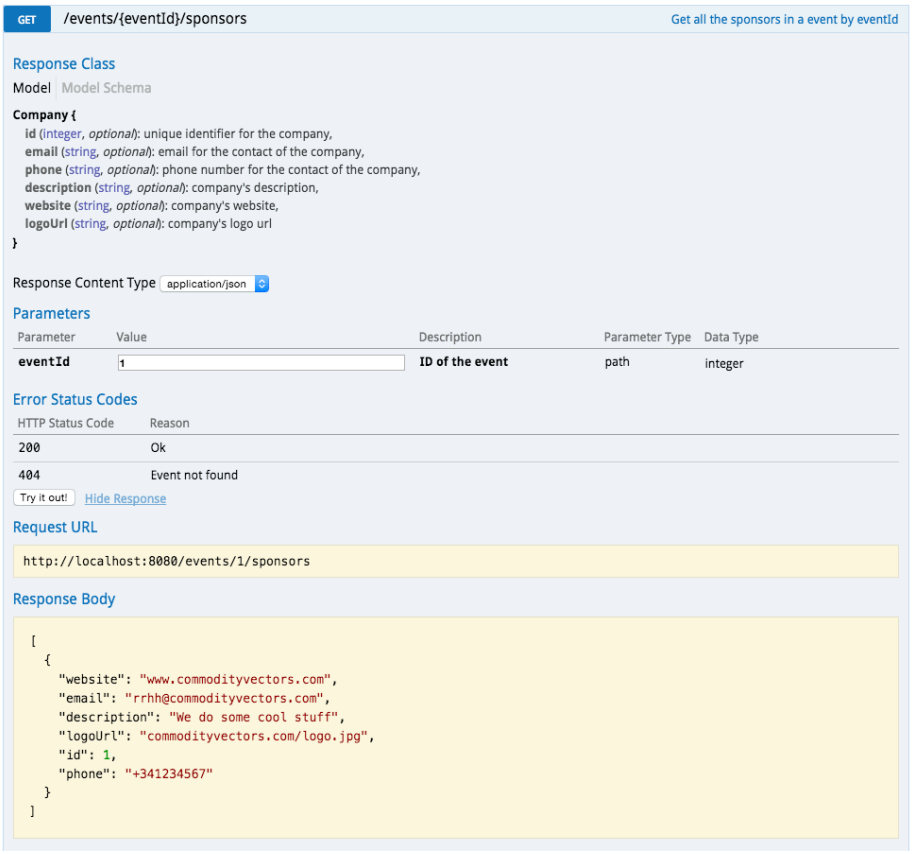


Figura 6 - "GET" sponsors para un evento

5. Eventos

Las operaciones básicas disponibles para los eventos (Figura 7 - Operaciones de eventos) son, crear eventos, obtener la lista de eventos, obtener un evento pasando un identificador y eliminar un evento.

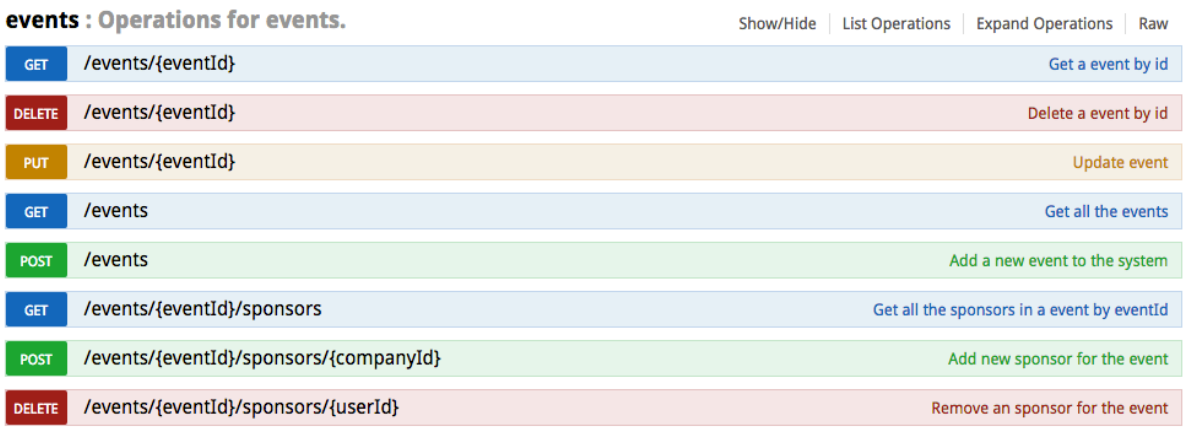


Figura 7 - Operaciones de eventos

También se han añadido los URI para obtener los sponsors de un evento así como agregar o eliminar un sponsor al evento. Para ello necesitamos que el sponsor sea una compañía (Figura 9 –) ya existente en el sistema.

5.1. Entidades relacionadas

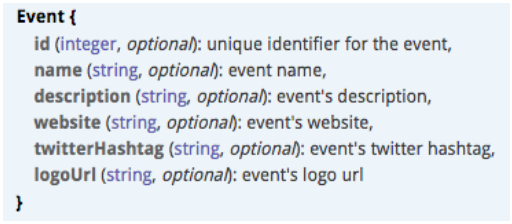


Figura 8 – Entidad Evento

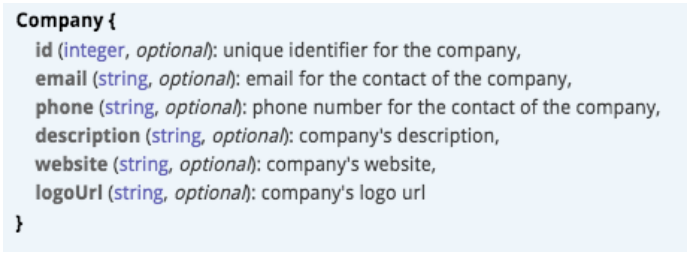


Figura 9 – Representación de Sponsor

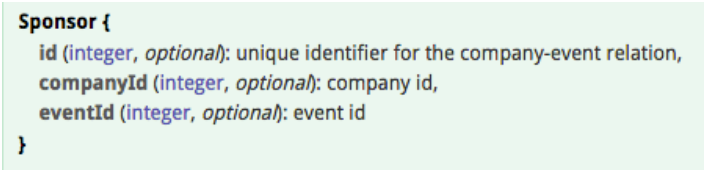


Figura 10 - Entidad Sponsor

Las figuras 8, 9 y 10, muestran las entidades que se obtienen en el endpoint de Eventos.

6. Usuarios

Las operaciones de usuarios son las básicas de una entidad. (Figura 11 - Operaciones de usuario)

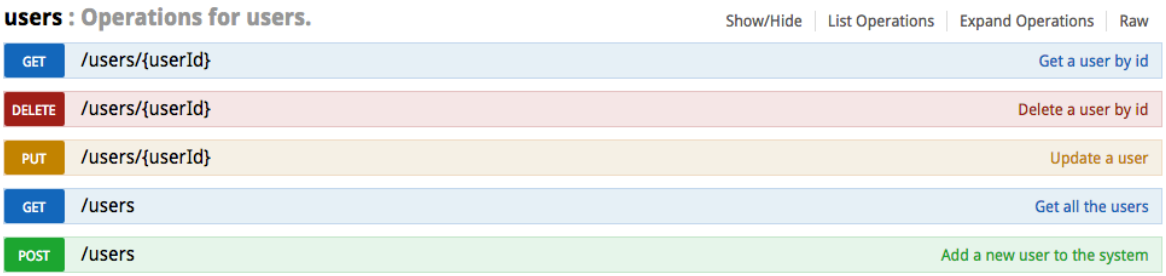


Figura 11 - Operaciones de usuario

La entidad para crear un usuario es UserDto (Figura 13 - Entidad UserDto), como se puede observar tiene ciertas diferencias con el User, los campos obligatorios para crear un usuario son: email, permission, y password. *Permission* es un valor string, no es una buena práctica pero nos permite completar la funcionalidad a la espera de más información por parte del cliente referente a permisos y roles (solo existe “ADMIN” por el momento).

6.1. Entidades relacionadas

User {
id (integer, *optional*): unique identifier for the user,
email (string, *optional*): email of the user,
firstName (string, *optional*): user's first name,
lastName (string, *optional*): user's last name,
twitterId (string, *optional*): user's twitter id,
linkedinId (string, *optional*): user's linkedin id,
bio (string, *optional*): user's bio,
permission (string, *optional*): user permission
}

Figura 12 – Entidad Usuario

UserDto {
email (string): email of the user,
firstName (string, *optional*): user's first name,
lastName (string, *optional*): user's last name,
twitterId (string, *optional*): user's twitter id,
linkedinId (string, *optional*): user's linkedin id,
bio (string, *optional*): user's bio,
permission (string): user permission,
password (string): password of the user
}

Figura 13 - Entidad UserDto

Las figuras 12 y 13 muestran las entidades que se pueden obtener en el endpoint de Usuarios.

7. Tipos de Actividad

Las operación de Tipo de actividad (Figura 14 – Operaciones de Tipos de actividad) son las básicas de un recurso. Para crear un nuevo tipo de actividad únicamente se necesita la descripción de esta (Figura 16 - Entidad ActivityTypeDto).

Figura 14 – Operaciones de Tipos de actividad

7.1. Entidades relacionadas

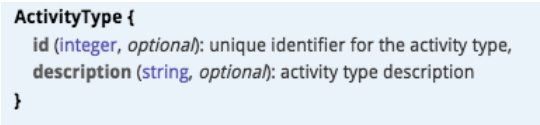


Figura 15 – Entidad Tipo de actividad

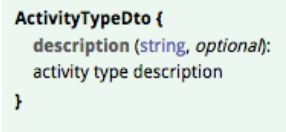


Figura 16 - Entidad ActivityTypeDto

Las figuras 15 y 16, muestran las entidades que se pueden obtener desde el endpoint Tipos de Actividad.

8. Compañías

Las operaciones del recurso “Company” (Figura 17 – Operaciones de Compañía) son las básicas. Los campos básicos para la creación de una compañía en el sistema son: nombre de la compañía y dirección de correo electrónico de contacto (Figura 19 - Entidad CompanyDto).

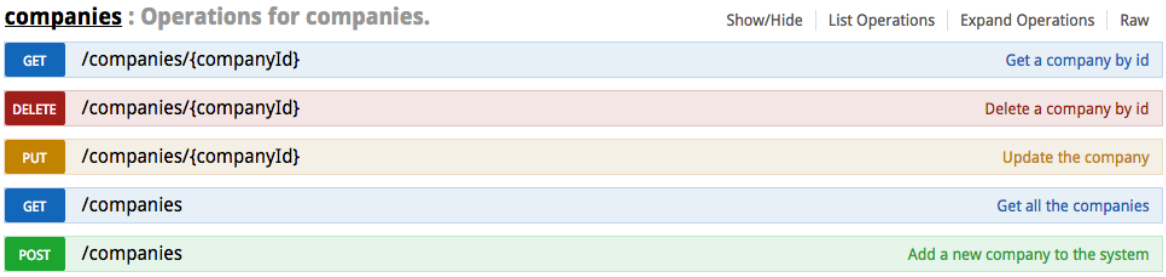


Figura 17 – Operaciones de Compañía

8.1. Entidades relacionadas

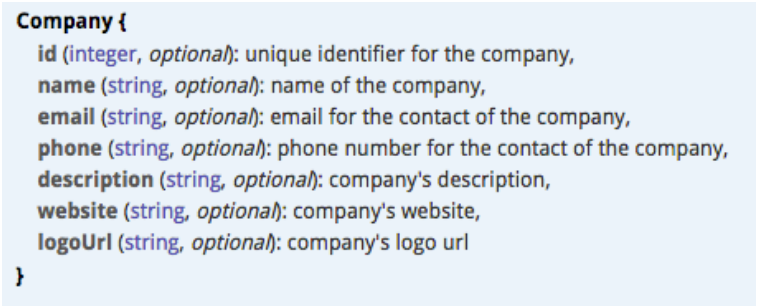


Figura 18 - Entidad de Compañía

Figura 19 - Entidad CompanyDto

Las figuras 18 y 19 muestran las entidades que se pueden obtener desde el endpoint de Compañía.

9. Días de evento

Las operaciones de días de evento son las básicas (Figura 20 – Operaciones Días de evento), los campos básicos para crear un día de evento son (Figura 22 - Entidad EventDayDto): id de evento, startTime (fecha y hora de comienzo), endTime (fecha y hora de final).

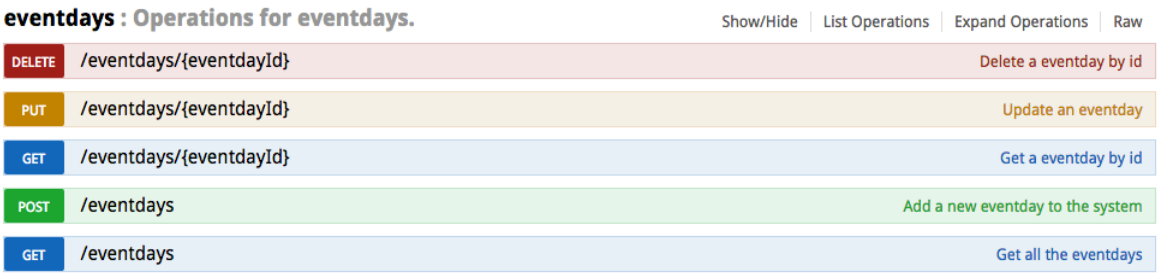


Figura 20 – Operaciones Días de evento

9.1. Entidades relacionadas

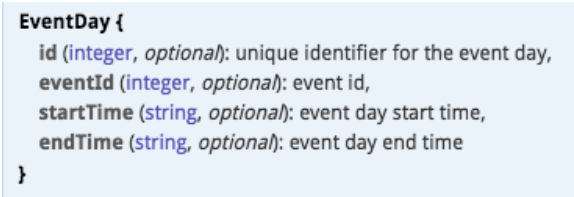


Figura 21 – Entidad Día de evento

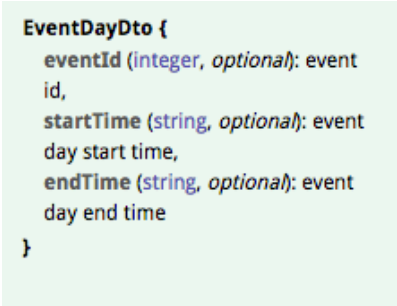


Figura 22 - Entidad EventDayDto

Las figuras 21 y 22 muestran los tipos de entidad que se pueden obtener desde el endpoint de Día de Evento.

10. Tag de Actividad

Las operación de tag de actividad son las básicas de un recurso (Figura 23 – Operaciones de Tag de actividad), el campo básico para crear un endpoint es el nombre, también existen otros campos opcionales como son *color* y *shortname (*nombre corto*)* *(*Figura 25 - Entidad TagDto*).*

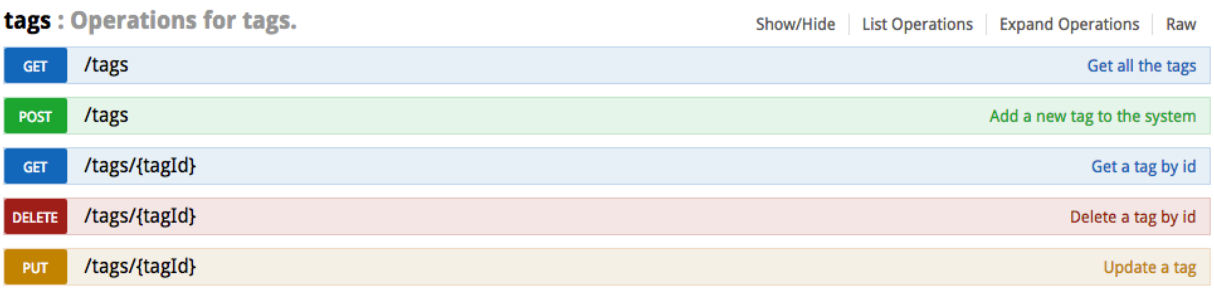


Figura 23 – Operaciones de Tag de actividad

10.1. Entidades relacionadas

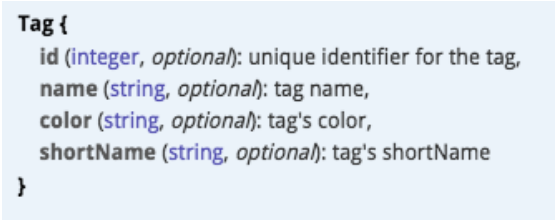


Figura 24 – Entidad Tag de actividad

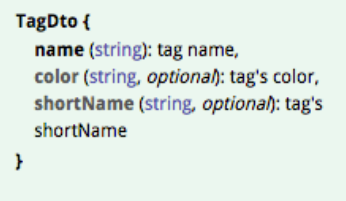


Figura 25 - Entidad TagDto

Las figuras 24 y 25 muestran los tipos de entidad que se pueden obtener desde el endpoint de Tag.

11. Localizaciones

Las operaciones básicas de localizaciones son las normales de un recurso (Figura 26 – Operaciones de Localizaciones), para crear una localización necesitamos como mínimo los siguientes campos: nombre, longitud, latitud y capacidad, los campos opcionales son (Figura 28 - Entidad LocationDto) : *code* (código), *description* (descripción) y *photoUrl* (url de una fotografía de la localización)

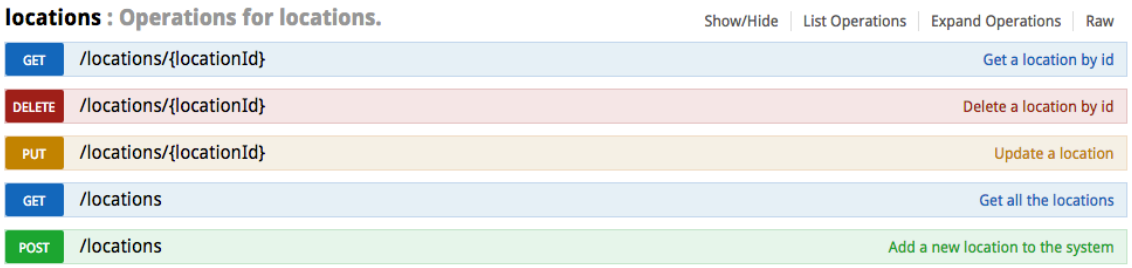


Figura 26 – Operaciones de Localizaciones

11.1. Entidades relacionadas

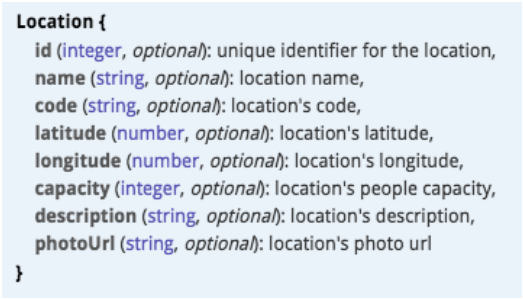


Figura 27 – Entidad Localización

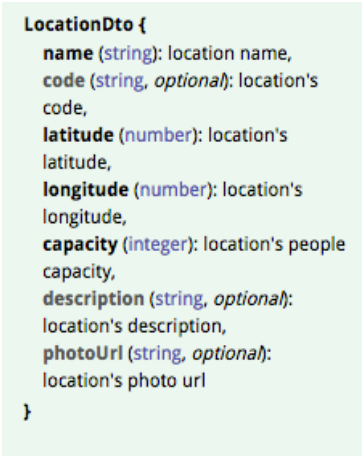


Figura 28 - Entidad LocationDto

Las figuras 27 y 28 muestran los tipos de entidad que se pueden obtener desde el endpoint de Localizaciones.

12. Actividades

El recurso actividad se podría denominar como el eje central de la aplicación (Figura 29 – Operaciones de Actividades), por una parte tenemos las acciones normales de un recurso, para crear una nueva actividad se necesitan los campos (Figura 31 - Entidad ActivityDto): eventId (Identificador del evento), locationId (identificador de la localizacion), activityTypeId (Identificador de la actividad), startTime (hora de comienzo), endTime (hora de fin).

Las operaciones especiales de actividad son las relacionadas con asistentes y ponentes de estas, podemos obtener la lista de asistentes/ponientes de una actividad, añadir un asistente/poniente o eliminar un asistente/poniente, únicamente necesitamos el identificador de usuario y el identificador de actividad.

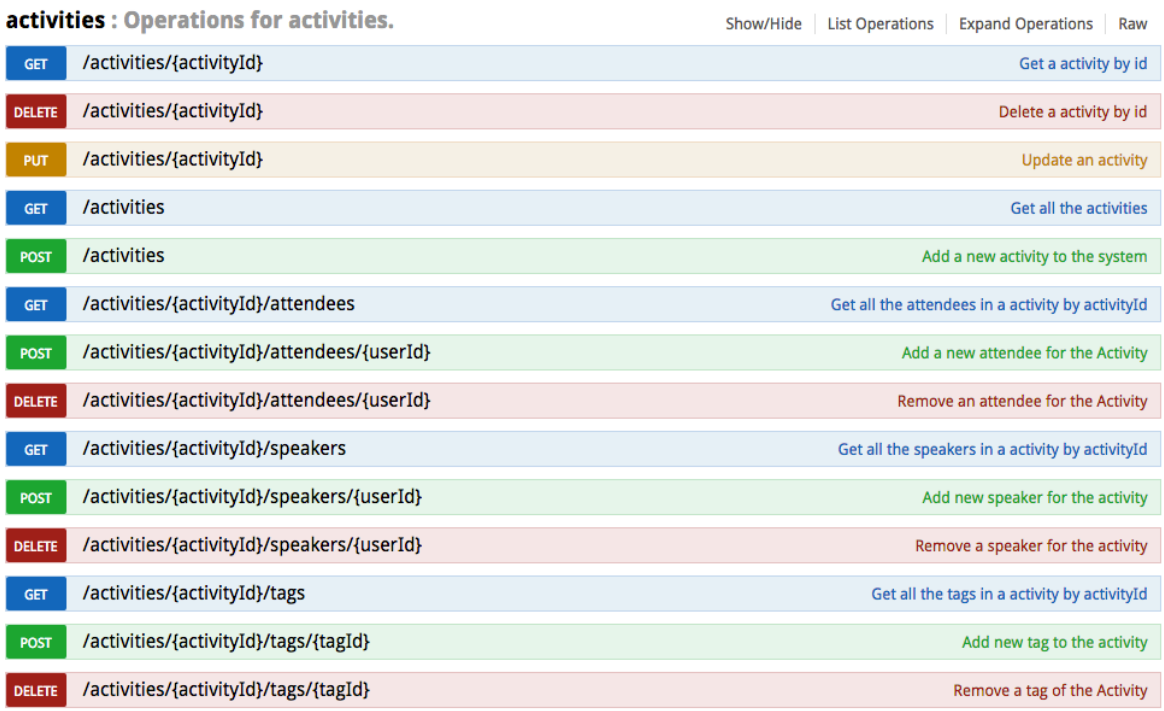


Figura 29 – Operaciones de Actividades

12.1. Entidades relacionadas

Activity {
id (integer, *optional*): unique identifier for the activity,
eventId (integer, *optional*): event id,
locationId (integer, *optional*): location id,
activityTypeId (integer, *optional*): activity type,
title (string, *optional*): activity's title,
description (string, *optional*): activity's description,
objective (string, *optional*): activity's objective in case of being a workshop,
startTime (string, *optional*): activity's start time,
endTime (string, *optional*): activity's end time,
resources (string, *optional*): resources may be a zip file with source code, links, pdf with instructions, powerpoint with slides, a link to youtube, etc (so, potentially anything)
}

Figura 30 – Entidad Actividad

ActivityDto {
eventId (integer): event id,
locationId (integer): location id,
activityTypeId (integer): activity type,
title (string, *optional*): activity's title,
description (string, *optional*): activity's description,
objective (string, *optional*): activity's objective in case of being a workshop,
startTime (string): activity's start time,
endTime (string): activity's end time,
resources (string, *optional*): resources may be a zip file with source code, links, pdf with instructions, powerpoint with slides, a link to youtube, etc (so, potentially anything)
}

Figura 31 - Entidad ActivityDto

Las figuras 30 y 31 muestran los tipos de entidad que se pueden obtener desde el endpoint de Actividades.

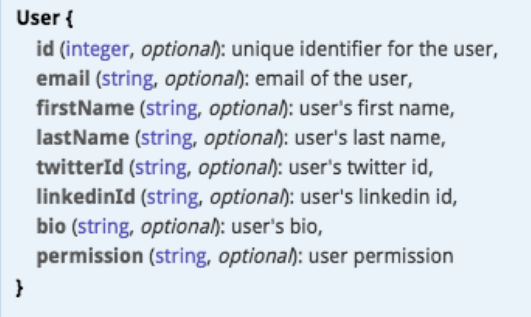


Figura 32 - Asistente/Poniente

13. Referencias

- Gonzalez, A. (s.f.). *Base64: la codificacion mas util en criptografia.* Obtenido de http://albertx.mx/blog/base64/
- W3.org. (s.f.). *Http 1.1*. Obtenido de http://www.w3.org/Protocols/rfc2616/rfc2616.html
- Wikipedia. (s.f.). *es.wikipedia.org*. Obtenido de https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme

Ingeniería Técnica Informática de Gestión

PROYECTO FIN DE CARRERA

ANEXO III : MANUAL DE DESPLIEGUE (ESTUDIO DEL STACK TECNOLÓGICO PARA EL DESARROLLO DE UN FRAMEWORK ESCALABLE ORIENTADO A APIS REST)

Autor: Giancarlo Alfredo Muñoz Reinoso
Tutor: Alejandro Baldominos Gómez

Leganés, Septiembre 2015

Tabla de contenidos

Índice de figuras

1. Introducción

En este documento vamos a describir el proceso que se lleva a cabo para desplegar la aplicación usando un servidor Heroku, el nuevo despliegue sería tan fácil como hacer un "*push*" al repositorio.

Heroku automáticamente haría el redespliegue por nosotros, este servicio es denominado *Platform-as-a-Service*. Heroku, como si de un miembro del equipo de devops se tratara, actualiza el código con la ultima versión del repositorio, crea el fichero desplegable (similar al .war) y se encarga de desplegarlo en el servidor, el despliegue en local es bastante sencillo también, simplemente lanzando el comando "sbt run".

Sin embargo vamos explicar paso a paso como se llevo a cabo esta integración de nuestro repositorio git con heroku.

1.1. Requisitos de Heroku

Heroku permite al desarrollador disponer de una potente herramienta en la que se despreocupa del mantenimiento y despliegue en el servidor, el desarrolldor únicamente se concentrará en el código.

La plataforma proporciona soporte a ciertos lenguajes de programación, afortunadamente para nosotros Scala es uno de ellos gracias sobre todo a la genial herramienta de compilación SBT.

Necesitamos crear una cuenta en Heroku, esto es sencillo simplemente accediendo a https://signup.heroku.com/dc

También es necesario tener instalado SBT http://www.scala-sbt.org/

2. Configuración

Se va a instalar una herramienta que nos proporciona Heroku llamada *"Heroku toolbelt"*, esta herramienta nos da acceso a la consola de Heroku dentro del bash, desde ahí se puede controlar la totalidad de la opciones de Heroku, desde configurar una base de datos para la aplicación, una herramienta de logging o una capa de cache de manera sencilla.

Tenemos distintas opciones respecto a sistema operativo para descargar *heroku-toolbelt*:

OSX:

Descargamos el archivo desde

Https://devcenter.heroku.com/toolbelt-downloads/osx

y seguimos las instrucciones.

Windows:

Descargamos el archivo desde

https://devcenter.heroku.com/toolbelt-downloads/windows

y seguimos las instrucciones

Debian/Ubuntu

Lanzamos el siguiente comando en la consola el siguiente comando

```
$ wget -0- https://toolbelt.heroku.com/install-ubuntu.sh | sh
```

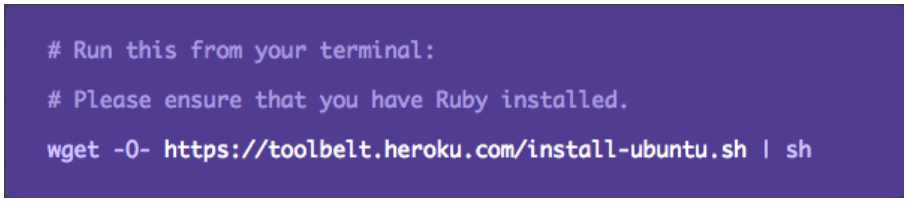


Figura 1 – Comando para instalación en Debian/Ubuntu

Una vez situados en la carpeta con el código clonado desde Github introducimos el comando

```
$ heroku create
```

Este comando prepara a Heroku para recibir el código de nuestra aplicación, este comando crea un git remoto y lo asocia a nuestro código. (Figura 2 - Creación App en Heroku)

Heroku genera un nombre aleatorio en nuestro caso “sleepy-shelf-2528”

Figura 2 - Creación App en Heroku

```
$ git push heroku master
Counting objects: 29, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (24/24), done.
Writing objects: 100% (29/29), 5.89 KiB | 0 bytes/s, done.
Total 29 (delta 0), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Play 2.x - Scala app detected
remote: -----> Installing OpenJDK 1.8... done
remote: -----> Priming Ivy cache (Scala-2.11, Play-2.3)... done
remote: -----> Running: sbt compile stage
remote:        Downloading sbt launcher for 0.13.8:
remote:          From  http://typesafe.artifactoryonline.com/typesafe/ivy-releases/org.scala-sl
remote:            To  /tmp/scala_buildpack_build_dir/.sbt_home/launchers/0.13.8/sbt-launch.ja
remote:        OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=384m; support was
remote:        Getting org.scala-sbt sbt 0.13.8 ...
remote:        downloading https://repo.typesafe.com/typesafe/ivy-releases/org.scala-sbt/sbt/0
remote:         [SUCCESSFUL ] org.scala-sbt#sbt;0.13.8!sbt.jar (85ms)
...
remote:        [info] Done packaging.
remote:        [success] Total time: 20 s, completed Apr 15, 2015 7:42:31 PM
remote: -----> Dropping ivy cache from the slug
remote: -----> Dropping sbt boot dir from the slug
remote: -----> Dropping compilation artifacts from the slug
remote: -----> Discovering process types
remote:        Procfile declares types          -> (none)
remote:        Default types for Play 2.x - Scala -> web
```

Figura 3 - "Push" del codigo a Heroku

3. Configuración de la base de datos

La base de datos se configura como un *plugin* dentro de la infraestructura de Heroku (https://addons.heroku.com/heroku-postgresql). Para instalarlo lanzamos el siguiente comando:

```
$ heroku addons:create heroku-postgresql
```

Nuestro código esta preparado para usar la variable de entorno DATABASE_URL donde se especifican las variables necesarias para conectar con la base de datos.

```
postgres://<username>:<password>@<host>/<dbname>
```

Figura 4 - Configuración de la base de datos

Vamos a mostrar una pequeña parte del código encargado de leer esta variable de entorno

```
12    val systemUrl: Option[String] = if (test) None else Option(System.getenv("DATABASE_URL"))
13
14    lazy val db: Database = systemUrl.fold[Database](Database.forURL(url, user, password, driver = driver)) {
15      uri =>
16
17        val systemUrl: Option[(String, String, String)] = for {
18          dbUri <- Option(new URI(uri))
19          userInfo = dbUri.getUserInfo().split(":").lift
20          user <- userInfo(0)
21          password <- userInfo(1)
22          host <- Option(dbUri.getHost)
23          port <- Option(dbUri.getPort)
24          databaseName <- Option(dbUri.getPath)
25        } yield (s"jdbc:postgresql://$host:$port$databaseName", user, password)
26
27        val tup = systemUrl.getOrElse(throw new Exception("FAILING GETTING DATABASE_URL"))
28
29        Database.forURL(tup._1, tup._2, tup._3, driver = driver)
30    }
```

Figura 5 - Código encargado de leer la variable de entorno

Esta es la única “adaptación” que tiene la aplicación en el código necesaria para ser lanzado en heroku con una base de datos proporcionada también por el sistema.

4. Script de base de datos

El script se encuentra también en :

https://github.com/Gneotux/pfc/blob/master/src/main/resources/schema.sql

Con este script accediendo a la base de datos mediante la consola de Postgres (PSQL) se podría hacer la inserción de las tablas simplemente copiando y pegando el contenido del fichero.

5. Proceso de despliegue

Cuando se halla completado el desarrollo, después de pasar las pruebas en local y se crea que la aplicación esta lista para pasar a producción únicamente se debe lanzar el comando:

```
$ git push heroku master
```

Con esto se envían los cambios al servidor. Al cabo de unos segundos cuando el comando se termina de ejecutar obtenemos una pantalla así (Figura 6 - Comando "git push heroku master"):

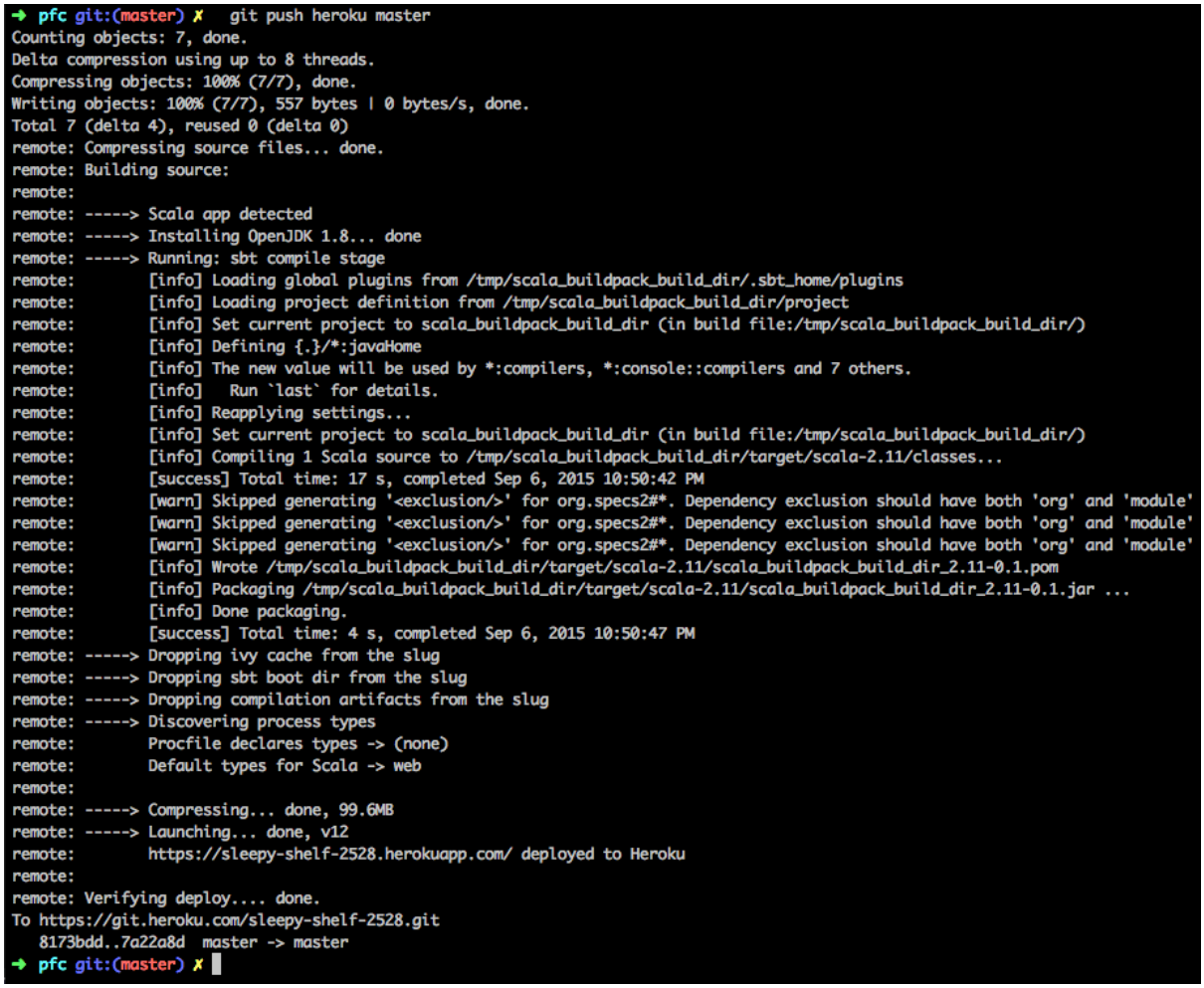


Figura 6 - Comando "git push heroku master"

Ahora podemos acceder:

https://sleepy-shelf-2528.herokuapp.com/